Казахский национальный университет имени аль-Фараби

УДК 004.4(043.4)

На правах рукописи

ҚАСЫМБЕК НҰРИСЛАМ МҰРАТБЕКҰЛЫ

Разработка эффективных параллельных алгоритмов в виде активных знаний для решения задачи неравновесной фильтрации(композиционная модель) и алгоритмов выполнения параллельных программ

8D06103 Компьютерные науки

Диссертация на соискание ученой степени
доктора философии (PhD)

Отечественный научный
консультанты: PhD, Лебедев Д.В.

Зарубежный научный консультант:
д.т.н., профессор Малышкин В.Э.
Новосибирск, РФ

Республика Казахстан
Алматы, 2025

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4
ВВЕДЕНИЕ 5
1 ОБЗОР СУЩЕСТВУЮЩИХ ЧИСЛЕННЫХ МЕТОДОВ И СРЕДСТВ ПАРАЛЛЕЛИЗАЦИИ РЕШЕНИЙ 9
1.1 Численные методы решения задач моделирования многофазных и многокомпонентных течений 9
1.1.1 Математические модели многофазных и многокомпонентных течений 9
1.1.2 Численные методы решения систем линейных алгебраических уравнений 11
1.1.3 Выводы 13
1.2 Архитектуры и средства параллельных вычислений 14
1.2.1 Обзор параллельных библиотек и инструментов 15
1.2.2 Параллельные численные методы 22
1.2.3 Алгоритмы параллельного исполнения метода GMRES 23
1.2.4 Выводы 24
1.3. Выводы по разделу 24
2 РАЗРАБОТКА АЛГОРИТМОВ ЧИСЛЕННОГО РЕШЕНИЯ ЗАДАЧИ НЕРАВНОВЕСНОЙ ФИЛЬТРАЦИИ (КОМПОЗИЦИОННАЯ МОДЕЛЬ) 25
2.1 Построение математической модели и выбор численного метода 25
2.2 Метод обобщённых минимальных невязок (GMRES) в составе итерационного алгоритма 29
2.3 Предобуславливание и его влияние на сходимость 31
2.4 Сходимость метода 35
3 РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ/РАСПРЕДЕЛЕННЫХ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ 37
3.1 Построение параллельного алгоритма метода GMRES на основе MPI 38
3.2 Разработка фрагментированного алгоритма численного решения 42
3.3 Сходимость фрагментированного алгоритма 48
3.4 Механизмы оптимизации 50
3.4.1 Уничтожающее потребление 50

3.4.2 Воспроизведение трасс 50
3.5 Включение алгоритма в базу активных знаний 51
4 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ АЛГОРИТМОВ 52
ЗАКЛЮЧЕНИЕ 60
Список литературы 63
ПРИЛОЖЕНИЕ А 69

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

GMRES – Generalized minimum residuals method
ILU – incomplete LU factorization
MPI – Message passing interface
LuNA – Language for Numerical Algorithms.
ФА – Фрагментированный алгоритм.
ФВ – фрагмент вычислений.
ФД – фрагмент данных.

ВВЕДЕНИЕ

Актуальность. В современном мире быстрыми темпами развиваются высокопроизводительные вычисления, что позволяет исследователям решать разные новые сложные научные и прикладные задачи, связанные с моделированием разных процессов, обработкой больших данных, и в целом, требующие большого объема вычислений. Одной из таких задач является решение систем дифференциальных уравнений численными методами. При решении систем дифференциальных уравнений численными методами, с увеличением требуемой точности решения возрастают и необходимые ресурсы для вычислений, в том числе и время вычисления. Расчет на суперкомпьютерах позволяет значительное ускорить решение многих проблем.

В области решения численных задач есть проблема разработки эффективных параллельных алгоритмов, запускаемых на суперкомпьютере. Под словом эффективность имеется в виду уменьшение времени выполнения и используемой памяти параллельных программ, динамическое распределение нагрузки между вычислительными узлами и т.д. Достижение хорошей эффективности параллельных программ может вызвать определенные трудности у разработчика параллельных программ. В этом направлении программист должен учитывать особенности выбранного численного метода, входных данных, вычислителя, распределения нагрузки между вычислительными узлами; реализовывать не только расчет, но и коммуникации между узлами. Сложность написания эффективных программ вызывает неоднородность, гетерогенность вычислителей (наличие GPU и других ускорителей).

Программы для высокопроизводительных вычислительных систем можно реализовать и вручную, и с использованием готовых решений. Одной из таких готовых решений является система LuNA, которая решает проблемы разработки эффективных параллельных программ.

LuNA – система фрагментированного программирования, которая дает возможность автоматизировать процесс параллельного исполнения разных программ на суперкомпьютере. Процесс достижения высшеперечисленных свойств автоматизирован в этой системе. Качество автоматически сгенерированных программ в определенных критериях могут уступать программам ручной реализации, но автоматизация позволяет уменьшить усилий на разработку, отладку и сопровождение кода.

Цели диссертационной работы разработка фрагментированного алгоритма для решения задачи моделирования течения многофазной многокомпонентной жидкости в виде, включаемом в базу активных знаний LuNA.

Задачи исследования:

1) Численное решение задачи моделирования многокомпонентного многофазного течения методами Ньютона и GMRES в задачах нефтедобычи;

2) Разработка фрагментированного алгоритма решения задачи нефтедобычи;

3) Реализация фрагментированного алгоритма с помощью MPI;

4) Реализация фрагментированного алгоритма на языке LuNA;

5) Улучшение разработанных алгоритмов и представление их в виде включаемом в систему LuNA

Объект исследования геофизические процессы в нефтяном пласте, высокопроизводительные вычисления, база активных знаний.

Предмет исследования параллельные и фрагментированные алгоритмы решения задачи неравновесной фильтрации.

Новизна полученных научных результатов

- впервые разработан фрагментированный алгоритм численного решения задачи моделирования течения многофазной многокомпонентной жидкости на основе метода Ньютона;

- разработана библиотечная подпрограмма, алгоритм с описанными свойствами для повторного использования при решении класса задач с системами уравнений, имеющих несимметричные и не диагонально доминирующие матрицы, допускающая динамическое управление, позволяющее динамически производить распределение ресурсов и балансировку нагрузки между вычислительными узлами.

Положения выносимые на защиту:

- Разработанные и реализованные крупноблочные фрагментированные алгоритмы для решения задачи моделирования течения многофазной многокомпонентной жидкости на основе метода Ньютона;

- Библиотечные подпрограммы с обеспечением реализации динамических свойств (управление ресурсами и вычислениями) для решения задачи фильтрации.

Теоретическая и практическая значимость исследования

Теоретическая значимость работы состоит в исследовании методов разработки параллельных/распределенных алгоритмов решения задач фильтрации с линеаризацией и дальнейшим решением линейных уравнений.

Практическая значимость работы состоит в возможности применения разработанных параллельных/распределенных алгоритмов для решения многих задач нефтедобычи, а также задач алгебры разреженных матриц. Разработанный фрагментированный алгоритм реализован в среде LuNA, что позволяет включить его в базу активных знаний, обеспечивая автоматическое, переносимое исполнение на различных вычислительных архитектурах. Это существенно снижает трудозатраты на разработку параллельных программ и открывает возможность многократного переиспользования алгоритма в задачах аналогичной структуры.

Достоверность научных результатов обеспечивается использованием апробированных методов (параллельного программирования, фрагментированного программирования, балансировки нагрузки), совпадением результатов вычисления различных параллельных алгоритмов на разнородных вычислительных ресурсах, внутренней непротиворечивостью предложенных алгоритмов, соответствием полученных результатов результатам других авторов

Связь темы с планами научно-исследовательских программ

Диссертационная работа выполнена в рамках научно-исследовательской работ:

- «Разработка системы управления активными знаниями для автоматизации конструирования высокопроизводительных параллельных программ обработки неструктурированных данных и численного моделирования в задачах фильтрации» - № гос. регистрации 0118РК00741, в ДГП НИИ математики и механики по грантовому финансированию, рассчитанной на 2018-2020 годы

- «Высокопроизводительное компьютерное моделирование движения многофазной жидкости в пористой среде в условиях неопределенности» - № гос. регистрации 0121РК00431, в РОО «НИА РК» по грантовому финансированию, рассчитанной на 2021-2023 годы

Структура и содержание диссертационной работы

Во введении обоснована актуальность темы диссертации, сформулированы цели и задачи исследования, научная новизна, методы исследования, практическая значимость работы, излагается краткое содержание глав диссертации.

В первой главе диссертации рассматриваются численные методы для решения задачи моделирования течения многокомпонентного многофазного течения, параллельные архитектуры, стандарты параллельного программирования и библиотеки параллельных подпрограмм. На основе результатов исследований были выбраны численные методы для решения рассматриваемой задачи и была выбрана фрагментированная технология для разработки параллельного алгоритма, что облегчает реализацию алгоритмов и производит автоматическую балансировку вычислительных процессов на вычислительной машине.

Во второй главе описывается математическая модель движения многокомпонентной многофазной жидкости в пористой среде. Рассматривается численное решение и приведена разностная схема одномерного случая, описан алгоритм решения линейных уравнений с большими разреженными матрицами.

В третьей главе описывается разработка фрагментированного алгоритма, реализация алгоритма в системе LuNA. Приведено описание фрагментированного алгоритма для решения выбранной задачи. Показаны пути сокращения времени выполнения фрагментированной программы.

В четвертой главе приводятся результаты тестирования с точки зрения эффективности реализованных параллельных алгоритмов. Проводится сравнение разработанных алгоритмов решения задачи нефтедобычи на основе метода Ньютона.

В заключении обобщаются основные теоретические и практические результаты, полученные в диссертационной работе, выделяются возможные направления дальнейших исследований и приводятся основные результаты данной диссертационной работы.

Апробация работы.

Результаты работы обсуждались на научных семинарах кафедры компьютерных наук и факультета информационных технологий Казахского Национального университета им. аль-Фараби, а также докладывались на следующих конференциях:

- IV Международная Научно-практическая конференции «Информатика и прикладная математика», (Алматы, 2019);
- 17th European Conference on the Mathematics of Oil Recovery (ECMOR XVII), (online, 2020);
- Традиционная международная апрельская математическая конференция в честь Дня работников науки Республики Казахстан, посвященная 75-летию академика НАН РК Тынысбека Шариповича Кальменова", (Алматы, 2021);
- The 18th European Conference on the Mathematics of Oil Recovery (ECMOR 2022), (Гаага, Нидерланды, 2022).

Публикации. По теме диссертации опубликованы 8 печатных работ, в том числе 2 в изданиях, рекомендованных КОКСНВО МНВО РК, 2 работ в рецензируемых журналах, входящих в международную базу цитирования «Web of Science» и/или «SCOPUS», 4 работы в сборниках международных конференций.

1 ОБЗОР СУЩЕСТВУЮЩИХ ЧИСЛЕННЫХ МЕТОДОВ И СРЕДСТВ ПАРАЛЛЕЛИЗАЦИИ РЕШЕНИЙ

В данной главе рассматриваются существующие численные методы, используемые для моделирования течения многофазной многокомпонентной жидкости в пористых средах, в том числе в задачах неравновесной фильтрации. Приводится сжатое изложение композиционных моделей, способов пространственно-временной дискретизации и методов численного решения соответствующих систем уравнений, включая итерационные алгоритмы и методы предобуславливания.

Особое внимание уделяется системам линейных алгебраических уравнений, возникающим при применении неявных схем аппроксимации, а также особенностям их численного решения. Рассматриваются подходы к реализации вычислительных алгоритмов на высокопроизводительных вычислительных системах, охватывающих архитектуры суперкомпьютеров, технологии параллельного программирования и библиотеки для поддержки параллельных вычислений.

Также представлены обзорные сведения по существующим параллельным алгоритмам решения неявных разностных схем, включая подходы, основанные на фрагментированной декомпозиции вычислений, что особенно актуально для задач, реализуемых в рамках систем активного знания.

1.1 Численные методы решения задач моделирования многофазных и многокомпонентных течений

1.1.1 Математические модели многофазных и многокомпонентных течений

В рамках работ была рассмотрена задача неравновесной фильтрации (композиционная модель). Задача моделирования течения жидкости в пористой среде является актуальной для нефтедобывающих стран и компаний. Существуют множество моделей для данной задачи, такие как модель Дарси, Баклея-Леверетта, Black-Oil. Первые два модели являются достаточно простыми и имеют ограничения и недостатки такие, как не учитывание инерционных эффектов, нелинейных эффектов, смешиваемости жидкости. Модель Black-Oil является более точной, но также упрощенный характер модели может не учитывать некоторые важные аспекты физико-химического поведения флюидов, особенно при значительных изменениях условий добычи. Нынешнее развитие вычислительных систем позволяет использовать композиционную модель, которая требует очень большие вычислительные ресурсы.

Композиционная модель. Для повышения точности моделирования сложных гидродинамических процессов в нефтяных пластах требуется учитывать компонентный состав каждой из фаз [1], [2]. Учет конвекции, диффузии, фазовых переходов и химических реакций затрудняет моделирование

подобных задач [3], [4], [5], [6]. Здесь можно рассмотреть композиционную модель. В данной модели учитываются детальный химический состав, фазовое равновесие, термодинамические свойства, процессы переноса и химические реакции.

Для решения дифференциальных уравнений, на которых основаны описанные модели, применяются численные методы. Численное решение дифференциальных уравнений включает в себя использование различных алгоритмов для приближенного решения уравнений, которые сложно или невозможно решить аналитически. Процесс решения таких уравнений часто требует дискретизации, то есть преобразования непрерывных функций и уравнений в дискретные формы. Это достигается путём разбиения времени и пространства на конечное число точек или интервалов и аппроксимации производных с помощью конечных разностей. К числу распространённых численных методов решения дифференциальных уравнений относятся метод Эйлера, метод Рунге–Кутты и метод конечных разностей.

Метод конечных объёмов базируется на принципе сохранения физических величин в пределах малых контрольных объёмов, на которые разбивается расчётная область. Он обеспечивает точное сохранение потоков и масс, что делает его особенно подходящим для решения задач с сохранением.

Метод конечных элементов применяется для решения задач в областях с комплексной геометрией и сложными граничными условиями. Он разбивает область на множество маленьких частей (элементов) различной формы и размера, что позволяет точно аппроксимировать геометрию исследуемого объекта.

Метод конечных разностей является одним из самых старых и наиболее широко используемых численных методов для решения дифференциальных уравнений. Он работает путем аппроксимации производных функций с помощью разностных уравнений на дискретной сетке.

Выбран метод конечных разностей (МКР) для решения дифференциальных уравнений из-за его простоты реализации и меньших требований к вычислительным ресурсам. Этот метод позволяет эффективно решать задачи на регулярных сетках, обеспечивая при этом достаточную точность для большинства инженерных и научных приложений.

Численные методы в зависимости от способа обработки временной компоненты уравнений, разделяются на явные и неявные. Для численного решения рассмотренных нами моделей существуют несколько подходов. В [7], [8] описаны основные три способа:

- неявный по давлениям, явный по насыщенности, IMPES (implicit in pressure and explicit in saturation);

- полностью неявный метод или метод одновременного решения, SS (simultaneous solution method);

- последовательный метод (sequential solution method).

Второй метод, решающий все связанные нелинейные уравнения одновременно, обеспечивает высокую устойчивость и точность, однако

характеризуется высокой вычислительной сложностью. Третий метод разбивает связанную систему нелинейных управляющих уравнений гидродинамического моделирования на отдельные уравнения и решает каждое из этих уравнений отдельно и неявно. Данный метод менее устойчив чем второй.

Другие методы можно увидеть в работах [9], [10], [11].

Выбор между явным и неявным методами зависит от свойств решаемой задачи. Явные методы обычно проще в реализации, но их устойчивость может зависеть от размера шага по времени. Неявные методы могут быть более устойчивыми, но их решение может потребовать более сложных численных методов, таких как методы типа Ньютона. Решение неявных схем приводит к решению систем линейных алгебраических уравнений вида

$$Ax = b \tag{1}$$

на каждом временном слое.

1.1.2 Численные методы решения систем линейных алгебраических уравнений

Есть различные методы решения систем вида (1) и у каждого метода есть свои преимущества и недостатки. При выборе метода решения СЛАУ важную роль играет размерность системы. Для решения СЛАУ маленьких размеров можно использовать формулы Крамера [12], но так как сложность данного метода O(n!), для решения практических задач он не используется. В основном методы решения СЛАУ можно разделить на две группы, прямые (в некоторых источниках точные) и итерационные методы.

К прямым методам относятся такие методы, как, метод Гаусса, разложение LU и т.д. Как работают прямые методы, в частности метод Гаусса и LU разложение описано в [12], [13]. Метод исключения Гаусса можно разделить на два этапа, прямой и обратный ход. При прямом ходе система поэтапно преобразовывается в треугольный вид, исключая на первом шаге неизвестный элемент x_1 из второго, третьего,..., n-ного уравнения, на втором шаге x_2 из третьего, четвертого, ..., n-ного уравнения и т.д. Данное преобразование позволяет при обратном ходе поэлементно найти неизвестные в обратном порядке. Похожим образом действует и метод LU разложения.

Также к прямым методам относится метод прогонки. Метод прогонки, также известный как алгоритм Томаса или метод Тридиагональной системы, применяется для эффективного решения систем линейных уравнений, в которых матрица системы обладает трехдиагональной структурой.

Основная идея метода заключается в том, чтобы привести исходную систему к трехдиагональному виду и затем эффективно решить получившуюся систему. Этот подход особенно полезен в случаях, когда имеется большое количество уравнений, и требуется оптимизированный вычислительный процесс.

Основные этапы метода прогонки включают в себя преобразование системы к трехдиагональному виду, прогонку для нахождения значений неизвестных на прямом ходе и обратную прогонку для окончательного получения решения. Этот метод демонстрирует высокую эффективность, особенно при решении задач с трехдиагональными матрицами, что делает его важным инструментом для численного решения различных задач, включая задачи на собственные значения и дифференциальные уравнения.

Как и в других уравнениях математической физики, система линейных уравнений, получаемая при решении задачи неравновесной фильтрации, является сильно разреженной. При использовании прямых методов для решения таких систем можно столкнуться с проблемой переполнения, то есть матрица преобразованной системы может хранить намного больше элементов чем исходная матрица. Для увеличения точности решения увеличивают и количество точек сетки, а это приводит к увеличению размера решаемой системы уравнений. Соответственно, размер матрицы и объем занимаемой ею памяти играют важную роль в компьютерных вычислениях. По этой причине для рассматриваемой задачи прямые методы не подходят.

К итерационным методам относятся метод Якоби, Зейделя, релаксация, CG, BiCG, GMRES и т.д. и их различные модификации. Итерационные методы находят приближенное решение СЛАУ повторяя определенные действия бесконечное количество раз [12]. Остановить повторения можно при достижении определенного максимального количества итераций или заданного условия сходимости.

Метод Якоби — это итерационный подход для нахождения решений систем линейных уравнений с симметричной и положительно определённой матрицей. Этот метод разлагает матрицу системы на диагональную и остальные компоненты, последовательно уточняя решение на каждой итерации.

В итерационных методах есть свои ограничения, например, в метод Якоби сходится в случае диагонального преобладания в матрице A. Метод Зейделя для нахождения неизвестных на определенном шаге использует найденные значения на том же шаге, что усложняет параллельную реализацию.

Для систем линейных уравнений с большими разреженными матрицами используют методы Крыловского типа, которые описаны в [14], [15]. Одним из таких методов является метод сопряженных градиентов, или CG (conjugate gradient method). Метод CG легко реализуем и содержит в основном матрично-векторные операции.

Метод сопряжённых градиентов — эффективный итерационный алгоритм для решения систем линейных уравнений с симметричными и положительно определёнными матрицами. Он минимизирует квадратичный функционал и улучшает решение на каждом шаге до достижения заданной точности, используя направления, сопряженные относительно матрицы системы.

Недостатком метода является ограничение в виде матрицы, а именно матрица системы должна быть симметрична. Для систем с несимметричными матрицами разработана модификация данного метода под названием метод

бисопряженных градиентов, или BiCG (Biconjugate gradient method). Но данный метод является не устойчивым, поэтому чаще используют метод BiCGStab (Biconjugate gradient stabilized method) или стабилизированный метод бисопряжённых градиентов[15]. В работе [16] автор разработал итерационный решатель ORTHOMIN для численного моделирования нефтяных пластов.

Для систем с большими разреженными матрицами общего вида широко используют метод обобщенных минимальных невязок, или GMRES (Generilized minimum residuals) [17]. Например, в работе [18] авторы использовали метод GMRES в моделировании многофазной жидкости и разработали предобуславливатель для метода. Преимущества метода GMRES относительно с другими итерационными методами, как устойчивость, критерий к виду системы, и его широкое использование в задачах моделирования нефтедобычи и не только, приводит нас к выбору данного метода как решатель СЛАУ.

В целях ускорения сходимости итерационных методов часто используются предобуславливание [14]. Предобуславливание – явная или неявная модификация исходной системы. Рассмотрим предобуславливание с помощью домножения исходной системы на некоторую матрицу M^{-1} , т.е. переход к системе вида

$$M^{-1}Ax = M^{-1}b \tag{2}$$

Матрица *M* называется предобуславливателем.

Часто используются предобуславливатели на основе LU разложения[19], [20]. В работах [21], [22] авторы разработали предобуславливатели на основе неполного LU, или ILU разложения. А в работе [23] разработали предобуславливатель ILU для ускорения метода GMRES.

1.1.3 Выводы

Показаны различные модели течения жидкости в пористых средах. Наиболее актуальным является композиционная модель, которая позволяет учесть детальный химический состав, фазовое равновесие, термодинамические свойства, процессы переноса и химические реакции. В работе будем применять численные методы для решения композиционной модели. Бóльшую точность можно достичь и выбором неявного метода для решения данной задачи, что в свою очередь приводит к решения СЛАУ.

Обзор методов показывает что существует множество известных методов решения систем алгебраических уравнений; прямые методы, такие как метод Гаусса или разложение LU, не подходят для систем с разреженными матрицами, поскольку существует возможность переполнения. Итерационные методы находят применение при решении задач крупного масштаба, когда использование прямых методов затруднено из-за высоких требований к объёму памяти и значительных временных затрат. Существует широкий спектр разновидностей итерационных алгоритмов, такие как CG, BiCG, GMRES и тд.

Ограничение метода CG его предназначение для систем с симметричными матрицами, а метод BiCG имеет медленную сходимость. Для систем с большими разреженными матрицами общего вида используется метод GMRES. Для ускорения итерационных методов применяется предобуславливание. Наиболее применимым методом для решения СЛАУ, необходимость которого возникает при неявном решении задачи неравновесной фильтрации, является метод GMRES с предобуславливанием ILU.

Достигаемая точность моделирования увеличивает вычислительную сложность, что в свою очередь увеличивает время вычислений. Данную проблему можно решить с помощью параллельных технологий.

1.2 Архитектуры и средства параллельных вычислений

Параллельные вычисления играют очень важную роль в современной науке. Развитие суперкомпьютеров позволяет решать все более объемные задачи. Важным критерием задач моделирования является время получение ответа, можно привести пример прогнозирования погоды и не только. Для разработки параллельных алгоритмов в первую очередь нужно разобраться с архитектурами параллельных систем и с существующими подходами и технологиями разработки параллельных программ.

Стандартной и известной классификацией параллельных архитектур является классификация Флинна. Данная классификация делит параллельные архитектуры в зависимости от количества потоков команд и данных. Из различных типов параллельных компьютеров нас интересуют мультикомпьютеры, мультипроцессоры и гибридные архитектуры, по которым конструируются кластерные системы, суперкомпьютеры.

Существуют много стандартов, библиотек для создания программ для мультикомпьютеров и мультипроцессоров. Для мультикомпьютеров, или систем с разделенной памятью, используется широко известный стандарт MPI [24]. А для мультипроцессоров, или систем с общей памятью, часто используют OpenMP [25]. Данные технологии часто используются при задачах моделировании, например в работе [26] MPI использовали при моделировании нефтеотдачи при заводнении полимером. А в работе [27], [28] авторы использовали OpenMP в моделировании движения нефти и газа. А в некоторых работах, например в [29] показано гибридное использование MPI и OpenMP. Вместе с этим, в современных суперкомпьютерах стали чаще использовать графический процессор – GPU(graphics processing unit). Для использования возможностей GPU, в программировании используют CUDA (Compute Unified Device Architecture) [30]. Данная технология позволяет существенно ускорить вычисления, например, в работах [31], [32] авторы использовали CUDA при моделировании движения многофазного потока.

Важной технологией в распределенных вычислениях является Грид. В грид-вычислениях для решения совместного выполнения общих задач используются часто географически распределенные, но соединенные сетями

несколько компьютеров [33], [34]. Отличием от классических суперкомпьютеров является то, что «виртуальный суперкомпьютер» в грид-вычислениях состоит из обычных компьютеров со стандартными комплектующими, соединенных с помощью обычных протоколов, а узлы в суперкомпьютере связаны через высокоскоростную сеть. Данная особенность означает что грид-вычисления лучше использовать для задач, в которых распределенные вычисления выполняются максимально автономно друг от друга, то есть при минимальном количестве коммуникаций.

При разработке параллельных программ можно опираться на описанные технологии. Но можно пойти иным путем и использовать готовые фреймворки, позволяющие использовать готовые решения для многих известных подзадач, которые возникают при решений задач математической геофизики. Например, матрично-векторные операции.

1.2.1 Обзор параллельных библиотек и инструментов

Существует ряд библиотек для решения задач линейной алгебры, такие как BLAS, LAPACK, PETSc, Charm++ и тд. Некоторые из них поддерживают параллельное выполнение в системах с распределенной или общей памятью.

BLAS

При рассматривании библиотек для решения задач линейной алгебры в первой очереди нужно остановиться на BLAS (Basic Linear Algebra Subprograms - базовые подпрограммы линейной алгебры)[35]. Это стандарт интерфейса программирования приложений для создания библиотек линейной алгебры. Функциональность BLAS можно поделить на три уровня:

Уровень 1: операция над векторами

Уровень 2: векторно-матричные операции

Уровень 3: операции типа матрица-матрица

Так как многие алгоритмы линейной алгебры состоят из подобных операций, многие библиотеки разработаны на основе данного стандарта.

LAPACK

Следующая библиотека – LAPACK (Linear Algebra PACKage) может решать основные задачи линейной алгебры и использует библиотеку BLAS [36]. Библиотека позволяет работать с плотными и ленточными матрицами. Изначально LAPACK был разработан чтобы эффективно работать на векторных и параллельных процессорах. Библиотека использует блочные операции работы над матрицами для использования преимуществ иерархического строения памяти в современных машинах. Подпрограммы LAPACK для решения разных задач вызывают подпрограммы BLAS, больше всего подпрограммы третьего уровня.

Существуют модификации библиотеки LAPACK для разных параллельных компьютеров. Один из которых – ScaLAPACK (Scalable Linear

Algebra PACKage). ScaLAPACK – является библиотекой высокопроизводительных подпрограмм линейной алгебры для систем с распределенной памятью. Библиотека также как LAPACK решает системы линейных уравнений с полными и ленточными матрицами, задачи наименьших квадратов, собственных значений и сингулярных значений. ScaLAPACK основан на идеях: блочное цикличное распределение матриц во время выполнения программы; расчеты выполняются на алгоритмах блочного распределения; хорошое моделирование низкоуровневых компонент, благодаря чему исходный код параллельной программы на высоком уровне не сильно отличается от последовательной программы. ScaLAPACK работает на машинах, где доступны BLAS, LAPACK, BLACS. Следующая библиотека PLASMA предназначена для решения задач плотной линейной алгебры с использованием OpenMP. PLASMA была разработана в целях заменить LAPACK и ScaLAPACK, но в данное время из за ограниченности функционала и работы только в системах с общей памятью не справляется с данной задачей. Еще одна библиотека под названием MAGMA решает идентичные задачи с LAPACK, но в основном предназначена для гетерогенных/гибридных архитектур. Библиотека состоит из параллельных алгоритмов для гибридных многоядерных систем с графическими ускорителями.

PETSc

PETSc (Portable, Extensible Toolkit for Scientific Computation) портативный, расширяемый инструментарий для научных вычислений[37]. Библиотека используется в больших прикладных проектах, соответственно, имеет возможность решить комплексную задачу согласованным набором компонентов. Библиотека содержит в себе множество решателей линейных и нелинейных систем, интеграторы обыкновенных дифференциальных уравнений. Кроме того, библиотека содержит множество механизмов поддержки параллельных программ, например, поддерживает распределенные параллельные матрицы, процедуры параллельного сбора матриц и векторов. Простой интерфейс является понятным и для новичков в области программирования, и автоматический реализует системную часть параллельных программ.

Charm++

Следующий фреймворк параллельного программирования Charm++, на языке C++ и адаптивной системой выполнения, позволяет писать параллельные программы в модулях без необходимости иметь дело с процессорами и потоками[38]. Особенность библиотеки, объектно-ориентированный дизайн программирования, то есть, алгоритмы реализуются с использованием объектов и вычисления происходят над взаимодействующими коллекциями объектов. Существенная разница от программировании с использованием стандарта MPI, возможность реализации глобально адресуемых объектов и переменных, к которым можно обращаться и которые могут быть вызваны из любых процессов. Размещение объектов на процессорах легко контролируем и есть возможность перенести любые объекты и переменные. Декомпозиция данных заключается в

разложении их в объекты одного класса, а параллельные вычисления над ними могут выполняться вызовом метода для всего набора объектов.

OpenCAEPoro

OpenCAEPoro — это открытый программный пакет для моделирования течения сжимаемой многофазной жидкости в пористых средах с механикой деформируемого скелета (coupled poromechanics)[39]. Разработка ориентирована на создание модульной и масштабируемой архитектуры, способной решать задачи, описываемые системой уравнений фильтрации Дарси, уравнениями механики сплошной среды и связями между ними. Платформа реализована на языке C++ с применением объектно-ориентированного и шаблонного программирования. OpenCAEPoro использует библиотеку PETSc для хранения матриц и векторов, а также для решения линейных и нелинейных систем уравнений. Для пространственной дискретизации применяется метод конечных элементов, что позволяет гибко описывать сложные геометрии и неоднородные свойства материалов. Основной акцент сделан на разбиение задачи на физические модули (фильтрация, механика, связка), которые могут независимо расширяться. Кроме того, в реализации используется подход «сборки по шаблону» (template-based assembly), что позволяет повторно использовать код при минимальных изменениях в модели. Параллельная реализация осуществляется на основе MPI, с возможностью масштабирования на большие кластеры. Однако в OpenCAEPoro отсутствует автоматическое управление распределением вычислений и балансировкой нагрузки: все аспекты параллелизма реализуются вручную на уровне MPI-кода. Это повышает требования к квалификации разработчика и усложняет перенос приложений на новые архитектуры.

LuNA

Использование перечисленных технологий и библиотек связано с необходимостью выполнения трудоёмких операций, таких как ручное распределение данных между процессами, задание схем обмена сообщениями между процессорами и указание передачи данных. Для автоматизации этих процедур существует система LuNA [40], обеспечивающая автоматическую генерацию параллельных программ и тем самым сокращающая объём работ, связанных с их разработкой [41]. Система LuNA, основанная на технологии фрагментированного программирования, опирается на вычислительную модель, называемую фрагментированной программой (ФП). В рамках данной модели данные задачи представляются совокупностью отдельных элементов — фрагментов данных (ФД), а процедуры — множеством фрагментов вычислений (ФВ). При разработке программы на языке LuNA необходимо задать зависимости между ФВ и ФД, то есть для каждого фрагмента вычислений определить, какие фрагменты данных используются в качестве входных и какие формируются на выходе. Система автоматически определяет зависимости,

конструирует параллельную программу, динамически балансирует вычисления и данные по вычислительным узлам.

Данная технология уже используется при решении таких задач, например в работе [42] систему LuNA использовали для трехмерного моделирования течения трехфазной жидкости. Авторам удалось получить ускорение относительно последовательной программы, но не относительно параллельной программы на MPI, так как система находится в стадии разработки.

Однако в ряде случаев возможно достижение высокой производительности даже с использованием автоматически сгенерированных программ. В работе [43] представлено расширение системы LuNA, специально ориентированное на поддержку операций плотной линейной алгебры. Авторы разработали специализированную подсистему выполнения, предназначенную для эффективного исполнения фрагментированных программ, описывающих алгоритмы таких разложений, как Cholesky, LU, LDL^T и других. Расширение включает компилятор, который определяет принадлежность входного алгоритма к поддерживаемому классу, и, при успешной идентификации, генерирует C++-код с применением специализированных алгоритмов планирования задач и распределения данных.

Предложенная система исполнения учитывает структуру задач и применяет блочно-циклическое распределение фрагментов данных, автоматически размещая фрагменты вычислений на соответствующих вычислительных узлах. Это позволило обеспечить эффективное выполнение параллельных программ, в том числе с учётом обмена сообщениями и синхронизацией зависимостей без участия программиста.

Проведённые авторами эксперименты по факторизации плотных матриц показали, что производительность сгенерированных программ в ряде случаев превосходит производительность реализации на базе библиотеки ScaLAPACK. Так, при выполнении алгоритма Cholesky на матрицах размером до 65536×65536 и при оптимальном размере блоков было получено ускорение до 2,4 раз. Эти результаты демонстрируют потенциал системы LuNA как инструмента для практического применения в задачах численного линейного алгебраического анализа, при условии наличия специализированной поддержки рассматриваемого класса задач.

Важной составляющей эффективной реализации фрагментированных программ в системе LuNA является стратегия размещения данных и вычислений на множестве вычислительных узлов. В работе [44] представлено исследование и реализация распределённого алгоритма размещения данных в системе LuNA, направленного на достижение динамической балансировки нагрузки и минимизации межузловых коммуникаций.

Авторы разработали два алгоритма — Hash-and-Track (HaT) и Rope-of-Beads (RoB), при этом основной акцент сделан на алгоритме RoB как более эффективном. Основная идея RoB заключается в проецировании многомерной структуры фрагментов данных на одномерный отрезок [0,1][0, 1][0,1] с помощью пространственно-заполняющей кривой (например, кривой Гильберта), что

позволяет сохранить локальность данных и эффективно их распределять между узлами. Коммуникации между процессами при этом происходят преимущественно локально, что особенно важно для масштабируемости на больших кластерах.

Проведённые численные эксперименты по решению уравнения Пуассона методом явной конечно-разностной схемы показали, что использование алгоритма RoB позволяет достичь лучших показателей производительности по сравнению с альтернативными подходами. При увеличении количества процессов (до 16) время исполнения существенно сокращается — до 15.2 секунд на тестовой задаче, что наглядно демонстрирует масштабируемость предложенного подхода.

Практическое применение системы LuNA для моделирования физических процессов демонстрируется, в частности, в работе [45], где рассмотрена эффективная реализация краевой задачи фильтрации двухфазной жидкости в проницаемой среде. Авторы разработали параллельную версию численного алгоритма на базе технологии фрагментированного программирования и реализовали её с использованием системы LuNA. В исследовании показано, что LuNA позволяет не только существенно упростить процесс параллельного программирования, но и обеспечить высокую эффективность при решении задач с нерегулярными сетками и сложной структурой вычислений.

Особое внимание в работе уделено сопоставлению производительности фрагментированной реализации с традиционными подходами. Авторы провели серию вычислительных экспериментов, продемонстрировавших масштабируемость и эффективность полученного решения при увеличении количества вычислительных узлов. Также подчёркивается, что применение LuNA позволило существенно упростить реализацию алгоритма на многопроцессорной архитектуре, так как вопросы распределения данных и управления зависимостями были автоматически решены системой.

Таким образом, статья подтверждает, что система LuNA применима не только для теоретических моделей и прототипов, но и для решения прикладных задач математической физики с реальными требованиями к производительности и надёжности.

Для изложения сути работы системы фрагментированного программирования LuNA нужно начать с понятия фрагментированный алгоритм. Фрагментированный алгоритм – это подход к программированию, который разбивает сложный вычислительный процесс на более мелкие и независимые фрагменты (или задачи), которые могут выполняться параллельно. Этот подход направлен на эффективное использование ресурсов параллельных вычислительных систем, таких, как многоядерные процессоры или кластеры.

ФД – фрагменты данных, части распределенных данных, в нашем случае матриц и векторов. Любые переменные, в понимании языка программирования, в системе LuNA являются ФД. Отличием ФД от переменных является их иммутабельность.

Программа для системы LuNA пишется на языке LuNA и представляет собой последовательность объявлений фрагментов кода. Фрагменты кода могут быть атомарными или структурированными.

Атомарные фрагменты кода имеют вид:

import <func_name> (<params>) as <code_id> ;

где func_name – имя подпрограммы, которая импортируется из файла на языке C/C++, params – входные и/или выходные параметры, code_id – имя фрагмента кода, по которому он будет доступен во фрагментированной программе.

Структурированный фрагмент кода имеет вид:

sub <code_id> (<params>) <body>

где body – тело, которое описывает множество операторов.

Один фрагмент кода задает один фрагмент вычислений – ФВ. ФВ принимает во вход 0 или более входных ФД и порождает 0 или более выходных ФД. Исполнение ФВ начинается как только все входные ФД готовы к вычислениям.

Предусмотрены операторы следующего вида:

- Оператор применения фрагмента кода:

  cf <cf_id>: <code_id> (<args_expressions>);

означает применение некоторого фрагмента code_id кода к конкретным аргументам args_expressions. cf_id – идентификатор фрагмента вычислений.

- Оператор внесения значения в контекст:

  let <name1>=<expr1>, <name2>=<expr2>, ... <body>

присваивает значение выражения exprX в параметр nameX, и параметры будут использованы в теле body.

- Оператор примитивно-рекурсивного перечисления:

  for <counter> = <first_expr> .. <last_expr> <body>

работает схожим образом с циклом с параметром, порождает множество фрагментов внутри тела body в зависимости от заданного выражения. Различие от традиционного цикла, порожденные фрагменты могут выполнятся не в порядке итерации counter, а порядок определяется только зависимостью данных.

- Оператор частично-рекурсивного перечисления:

  while <cond_expr> , <counter> = <start_expr> .. out <out_df_id> <body>

так же как предыдущий пример.

- Условный оператор:

  if <cond_expr> <body>

аналог условного оператора традиционных языков программирования.

Фрагментированная программа (ФП) состоит из ФВ, каждый из которых является отдельным процессом при исполнении. ФВ выполняются асинхронно, что позволяет эффективно использовать все доступные вычислительные ресурсы. Каждый процесс может выполняться на отдельном ядре процессора или на отдельном узле вычислительного кластера. Система LuNA выполняет перенос необходимых ФД между узлами. Выполнение ФП можно представить в виде ориентированного графа, вершинами которого являются ФД и ФВ. Задачей программиста является описание на языке LuNA зависимостей (ребер) между

фрагментами, последовательность выполнения процессов определяется run-time системой LuNA.

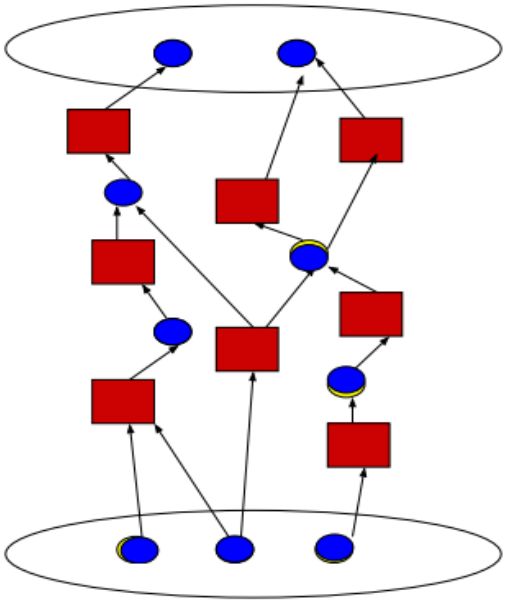


Рисунок 1 – Граф фрагментов (синие круги – ФД, красные прямоугольники – ФВ)

В отличии от MPI, в котором каждый процесс выполняет одну строку кода для своих данных, на языке LuNA пишутся отдельные фрагменты кода и выполняются отдельные ФВ для каждого ФД. При выполнении одинаковых операций на нескольких данных, для порождения множества схожих ФВ поможет Оператор примитивно-рекурсивного перечисления. Например:

```
for i=0..$FG_COUNT-1
{
        cf ini[i]: init($RESTR, m[i]);
}
```

данные строки кода показывают инициализацию ФД m[i] атомарным фрагментом кода init. ФВ ini[i] независимый друг от друга, и как было сказано ранее, for не определяет порядок их вычислений, ФВ ini[$FG_COUNT-1] может быть выполнена ранее ФВ ini[0].

Размер ФД не изменяется при выполнении ФП и определяется пользователем. Размер ФД сильно влияет на производительность программы. Слишком большой размер может не поместиться в память одного узла, и маленьким количеством ФВ нельзя достичь максимальной параллелизации. Слишком маленький размер ФД приводит к росту количества фрагментов, и приводит к дополнительным затратам при реализации управления.

Отличие и преимущество системы LuNA в том, что нет необходимости реализации коммуникаций, синхронизации, распределении задачи и данных

между процессами, балансировки нагрузки. Система LuNA автоматический выполняет эти работы, системой реализуется распределение фрагментов, обмен нужных фрагментов между узлами, синхронизация потоков, балансировка фрагментов между узлами. Важным преимуществом является независимость сгенерированной программы от архитектуры вычислителя. Единожды написанная фрагментированная программа может быть скомпилирована для различных вычислителей с различной структурой и топологией сети.

Еще одним преимуществом системы, является база активных хнаний. База активных знаний LuNA представляет собой библиотеку подпрограмм, но с одной важной особенностью: эти подпрограммы могут быть автоматически выбраны и использованы для решения конкретных задач. В отличие от традиционных библиотек, где функции и методы вызываются явно, в базе активных знаний LuNA реализация различных методов и функций написана на языке фрагментированного программирования LuNA, что позволяет системе самостоятельно выбирать и адаптировать необходимые компоненты для решения задач.

Одним из ключевых преимуществ базы активных знаний LuNA является её адаптивность. Программы, написанные сегодня, могут быть перекомпилированы и оптимизированы для будущих архитектур суперкомпьютеров. Это означает, что система LuNA будет продолжать улучшаться, обеспечивая высокую производительность и эффективность на новых платформах.

Актуальной задачей является наполнение базы активных знаний LuNA новыми методами и функциями. Это позволит в будущем решать конкретные задачи, используя уже реализованные и оптимизированные методы. Важно, чтобы база активно пополнялась и обновлялась, обеспечивая доступ к передовым алгоритмам и подходам для решения широкого спектра задач.

1.2.2 Параллельные численные методы

Параллельные вычисления играют важную роль при решений различных задач нефтяной геофизики, математической геофизики. Приведенные ранее численные методы при увеличении размеров разностной сетки требуют большое количество вычислительной времени. Как и было сказано ранее, в неявных методах решение системы приводится к решению линейных уравнений. Как показано в работе[46], 60% вычислительного времени занимает решение этих систем линейных уравнений. Поэтому во многих работах рассматривается разработка параллельных линейных решателей.

В работе [46] авторы разработали параллельный линейный решатель на основе метода обобщенных сопряженных остатков и Ортомин. Авторами была показана сложность разработки таких параллельных алгоритмов, как результат, было получено ускорение в 1.7 раз на 64 процессах и 3.8 раз на 128 процессах. Результаты показывают линейное ускорение относительно параллельных запусков на разном количестве процессов.

В многих работах можно увидеть что в неявных схемах используют метод прогонки. Известным примером параллелизация данного метода является метод предложенный Н.Н. Яненко[47], который уменьшает исходную систему с множеством неизвестных до системы с числом неизвестных, соответствующим числу процессоров. Такая система, представленная параметрическими граничными точками, решается с использованием метода прогонки. В зарубежных источниках метод прогонки более известен как алгоритм Томаса. В работе [48] авторы разработали параллельный алгоритм swPTS для многоядерных процессоров sw26010pro. Авторами было получено ускорение в 5.74 раз по сравнению с последовательным алгоритмом.

Как указано ранее для решения СЛАУ возникаемых при решении задачи неравновесной фильтрации лучше всего подходит метод GMRES. Поэтому далее приведены работы с параллельной реализацией данного метода.

1.2.3 Алгоритмы параллельного исполнения метода GMRES

Параллельные вычисления часто используются при решении систем линейных уравнений методами Крыловского типа. К примеру, в работах [49], [50] авторы схожим образом реализовали параллельную версию метода GMRES. В этих работах для построения ортогонального базиса был использован метод Ньютона. А в работе [51] авторы также использует базис Ньютона, но с предобуславливателем Шварца.

В следующей работе [52] авторы для уменьшения потери времени на ожидание глобальных коммуникации использовали асинхронные и неблокирующие операции коммуникации, на фоне которых процессоры могут производить другие вычисления. В результате был разработан конвейерный алгоритм GMRES.

В работе [53] предложен модифицированный блочный метод GMRES с дефляцией на каждой итерации для решения систем с несколькими правыми частями. Авторы также рассмотрели переменное предобуславливание и техники усечения подпространства, позволяющие сократить вычислительные затраты. Метод продемонстрировал эффективность на задачах геофизического моделирования, обеспечив до 35% ускорения по сравнению с классическим блочным GMRES.

В работе [54] авторы разработали параллельный алгоритм GMRES для кластерных систем с несколькими графическими процессорами. В предложенным авторами подходе общие вычисления, требующие коммуникаций, выполняются на CPU, а локальные вычисления выполняются на GPU, за счёт чего было достигнуто ускорение.

В работе [55] рассматриваются способы ускорения решения СЛАУ в задачах моделирования нефтяных пластов с помощью графических процессоров. Авторами реализованы предобусловленные решатели GMRES и AMG на многографических системах. Для повышения эффективности использованы ILU(k), RAS и специальные схемы распределения матриц. Получено до 35-

кратного ускорения на четырёх GPU при решении системы из модели SPE10, что подтверждает перспективность GPU в задачах фильтрации.

1.2.4 Выводы

Рассмотрены стандарты и библиотеки параллельных программ. Разработка параллельных алгоритмов и их реализация при применении многих из библиотек, а особенно стандартов как MPI, требует полного участия программиста при декомпозиции данных, пересылке нужных сообщений. Система LuNA автоматически реализует разделение вычислений и данных по узлам параллельной системы. К тому же, производит динамическую балансировку фрагментов между узлами. Предполагается разработка фрагментированного алгоритма метода GMRES для системы LuNA.

1.3. Выводы по разделу

Рассмотрены численные методы и технологии параллелизации. По причине устойчивости и точности был выбран полностью неявный метод. Для решения линейных уравнений был выбран метод GMRES из-за устойчивости и меньших требований к виду матрицы коэффициентов СЛАУ. Выбранный численный метод является ресурсоемким, что подталкивает к использованию параллельных вычислений.

Были рассмотрены разные библиотеки параллельных вычислений. Использование многих библиотек требуют существенных усилий со строны программиста. Например, распределение данных/вычислений по параллельным процессам. Для реализации алгоритмов была выбрана система LuNA, так как она дает возможность автоматического обеспечения динамических свойств выполнения параллельных программ. Также были рассмотрены параллельные решения для систем с различными архитектурами. Еще одним преимуществом системы LuNA является ориентированность на различные архитектуры.

2 РАЗРАБОТКА АЛГОРИТМОВ ЧИСЛЕННОГО РЕШЕНИЯ ЗАДАЧИ НЕРАВНОВЕСНОЙ ФИЛЬТРАЦИИ (КОМПОЗИЦИОННАЯ МОДЕЛЬ).

2.1 Построение математической модели и выбор численного метода

Рассматривается математическая модель многофазного и многокомпонентного течения в пористых средах, позволяющая описывать современные технологии добычи углеводородов, включая закачку теплоносителя в пласт, процессы внутрипластового горения, а также закачку химических реагентов (полимеров, поверхностно-активных веществ и др.).

Далее рассматривается математическая постановка задачи для трёхфазной многокомпонентной жидкости. В основе лежит закон сохранения массы, записанный для каждого компонента в отдельности:

$$\frac{\partial(\phi\,\xi_\omega S_\omega)}{\partial t} + \nabla\cdot(\xi_\omega \boldsymbol{u}_\omega) = q_\omega,$$

$$\frac{\partial(\phi[x_{mo}\xi_o S_o + x_{mg}\xi_g S_g])}{\partial t} + \nabla\cdot\left(x_{mo}\xi_o\boldsymbol{u}_o + x_{mg}\xi_g\boldsymbol{u}_g\right) = q_m \quad (3)$$

$$\boldsymbol{u}_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\boldsymbol{k}(\nabla p_\alpha - \rho_\alpha \wp \nabla z), \qquad \alpha = \omega, o, g. \quad (4)$$

$$\sum_{m=1}^{N_c} x_{mo} = 1, \qquad \sum_{m=1}^{N_c} x_{mg} = 1. \quad (5)$$

$$S_\omega + S_o + S_g = 1. \quad (6)$$

$$p_{co\omega} = p_o - p_\omega\,,\; p_{cgo} = p_g - p_o. \quad (7)$$

$$f_{mo}\left(p_o, x_{1o}, x_{2o,}\ldots, x_{N_c o}\right) = f_{mg}\left(p_g, x_{1g}, x_{2g}, \ldots, x_{N_c g}\right), \quad (8)$$

m=1.2,...,N_c, где ϕ – пористость, ξ_w – молярная плотность водной фазы, S_w, S_o, S_g – насыщенность водной, нефтяной и газовой фаз, q_w, q_m – молярные скорости потока воды и m-й компонент соответственно, $x_{i\alpha}$ - мольная доля, $k_{r\alpha}$ – относительная фазовая проницаемость фазы α, μ_α – вязкость фазы α, $\boldsymbol{k}$ – абсолютная проницаемость, $\wp$ - гравитационное ускорение, p_α – давление фазы α, $p_{co\omega}$ и p_{cgo} – капиллярные давления, f_{mo} и f_{mg} функции летучести, N_c – количество компонентов.

Существуют несколько математических методов управления поведением углеводородов (распределением химических компонентов по фазам). Наиболее распространенные основаны на:

- подходе значений K;
- уравнениях состояния (EOS);
- множестве эмпирических таблиц, полученных из экспериментов.

В этой работе мы использовали первые два метода.

Распределение химических компонентов в углеводородной фазе описывается методом K-значения. Равновесный коэффициент мгновенного испарения для каждой компоненты задается в следующем виде:

$$K_m = x_{mg}/x_{mo}\,,\ m = 1,2,\dots,N_c \tag{9}$$

где K_m – равновесное K-значение компоненты m.

Для более точного предсказания составов, плотностей и молярных объемов широко используются уравнения состояния (EOS – equation of state). Термодинамическое поведение жидкостей в пластовых условиях описывается уравнениями состояния Пенга-Робинсона. Летучесть компоненты m задается в следующем виде:

$$f_{m\alpha} = p_\alpha x_{m\alpha}\varphi_{m\alpha}, m = 1,2,\dots,N_c, \alpha = o,g. \tag{10}$$

где $\varphi_{m\alpha}$- коэффициент летучести компонента m.

Для улучшения вычислительной эффективности уравнения (2) - (8) можно переписать, используя первичные переменные. Далее мы используем общую массовую переменную:

$$F = \xi_o S_o + \xi_g S_g \tag{11}$$

и массовые доли:

$$L = \frac{\xi_o S_o}{F}, V = \frac{\xi_g S_g}{F}. \tag{12}$$

При этом, отметим, что:

$$L + V = 1$$

Далее, вместо использования отдельных мольных долей, мы используем общую мольную долю компонентов в углеводородной системе:

$$z_m = Lx_{mo} + (1 - L)\,x_{mg},\ m = 1,2,\dots,N_c \tag{13}$$

Используя (5), (11) и (12) увидим, что:

$$\sum_{m=1}^{N_c} z_m = 1 \tag{14}$$

Введя уравнение (2) во второе уравнение в (1), получим:

$$\frac{\partial(\phi F z_m)}{\partial t} - \nabla\cdot\left(\boldsymbol{k}\left[\frac{x_{mo}\xi_o k_{ro}}{\mu_o}\left(\nabla p_o - \gamma_o \nabla z\right) + \frac{x_{mg}\xi_g k_{rg}}{\mu_g}\left(\nabla p_g - \gamma_g \nabla z\right)\right]\right) = x_{mo}q_o + x_{mg}q_g, m = 1,2,\dots,N_c, \tag{15}$$

где $\gamma_\alpha = \rho_\alpha g, \alpha = o,g$. Учитывая (5) и (14) перейдем к следующему виду:

$$\frac{\partial(\phi F)}{\partial t} - \nabla \cdot \left(\boldsymbol{k} \left[\frac{\xi_o k_{ro}}{\mu_o} \left(\nabla p_o - \gamma_o \nabla z \right) + \frac{\xi_g k_{rg}}{\mu_g} \left(\nabla p_g - \gamma_g \nabla z \right) \right] \right) = q_o + q_g \quad (16)$$

Для упрощения уравнений определим проводимости:

$$\boldsymbol{T}_\alpha = \frac{\xi_\alpha k_\alpha}{\mu_\alpha} \boldsymbol{k}, \alpha = \omega, o, g$$
$$\boldsymbol{T}_{m\alpha} = \frac{x_{m\alpha} \xi_\alpha k_\alpha}{\mu_\alpha} \boldsymbol{k}, \alpha = o, g, m = 1,2, \dots, N_c \quad (17)$$

Равновесное соотношение (6) преобразуется в следующий вид:

$$f_{mo}\left(p_o, x_{1o}, x_{2o,} \dots, x_{N_c o}\right) = f_{mg}\left(p_o + p_{cg}, x_{1g}, x_{2g}, \dots, x_{N_c g}\right), \quad m = 1,2, \dots, N_c \quad (18)$$

Далее подставляя (17) в (15), получим уравнения следующего вида:

$$\frac{\partial(\phi F \chi_m)}{\partial t} = \nabla \cdot \left(\boldsymbol{T}_{mo} \left(\nabla p_o - \gamma_o \nabla z \right) + \boldsymbol{T}_{mg} \left(\nabla p_g - \gamma_g \nabla z \right) \right) + x_{mo} q_o + x_{mg} q_g, m = 1,2,\dots,N_c - 1 \quad (19)$$

Также для (16):

$$\frac{\partial(\phi F)}{\partial t} = \nabla \cdot \left(\boldsymbol{T}_o \left(\nabla p_o - \gamma_o \nabla z \right) + \boldsymbol{T}_g \left(\nabla p_g - \gamma_g \nabla z \right) \right) + q_o + q_g \quad (20)$$

Также поставим (17) в первое уравнения из (3):

$$\frac{\partial(\phi \, \xi_\omega S_\omega)}{\partial t} = \nabla \cdot \left(\boldsymbol{T}_w \left(\nabla p_w - \gamma_w \nabla z \right) \right) + q_\omega \quad (21)$$

В конце, поставив (11) и (12) в (6) получаем уравнение:

$$F \left(\frac{L}{\xi_o} + \frac{1 - L}{\xi_g} \right) + S = 1 \quad (22)$$

В итоге получаем преобразованную систему (18)-(22) с неизвестными: $x_{mo}, L, F, \chi_m, S = S_w, p = p_o, m = 1,2, \dots, N_c - 1$. Данная система уравнений является нелинейной.

Одним из методов решения нелинейных систем является метод Ньютона. Метод Ньютона основан на линеаризации исходной системы на каждом шаге итерационного процесса. Пусть на l-й итерации на временном шаге $n + 1$ имеется приближение $x^{n+1,l}$, представляющее вектор всех искомых переменных. Обновление на следующей итерации осуществляется по формуле:

$x^{n+1,l+1} = x^{n+1,l} + \triangle x^{n+1,l+1}$,

где приращение $\triangle x^{n+1,l+1}$ определяется из линеаризованной системы:

$$J(x^{n+1,l}) * \triangle x^{n+1,l+1} = -F(\triangle x^{n+1,l}) \quad (23)$$

где $F(x)$ – вектор функций (правая часть исходной нелинейной системы), а $J(x)$ – матрица Якоби, состоящая из частных производных компонент F по компонентам x:

$$J_{ij} = \frac{\partial F_i}{\partial x_j}$$

Для одномерной задачи и дискретизации по сетке, в каждой точке сетки формируются уравнения, содержащие зависимости от переменных данной и соседних ячеек. Это приводит к системе уравнений блочной структуры, где каждый блок матрицы Якоби соответствует производным по переменным в конкретной ячейке и соседних. Таким образом, результирующая матрица системы имеет разреженную блочную структуру, где каждый блок отражает физические и численные связи между переменными.

Система нелинейных уравнений (18) - (22) была линеаризована данным методом.

В (l+1)-м слое итерационного процесса Ньютона значение неизвестных обновляется по следующему закону [56]:

$$\begin{gathered} x_{mo}^{n+1,l+1} = x_{mo}^{n+1,l} + \triangle x_{mo}^{n+1,l+1},\ m = 1,2,\dots,N_c - 1, \\ \zeta_{mo}^{n+1,l+1} = \zeta_{mo}^{n+1,l} + \triangle \zeta_{mo}^{n+1,l+1}, \qquad m = 1,2,\dots,N_c - 1, \\ L^{n+1,l+1} = L^{n+1,l} + \triangle L^{n+1,l+1}, \\ F^{n+1,l+1} = F^{n+1,l} + \triangle F^{n+1,l+1}, \\ S^{n+1,l+1} = S^{n+1,l} + \triangle S^{n+1,l+1}, \\ p^{n+1,l+1} = p^{n+1,l} + \triangle p^{n+1,l+1}. \end{gathered} \quad (24)$$

Далее для одномерной вычислительной сетки, для каждой точки записываем уравнения (24) и получаем блочную матрицу, которая является матрицей коэффициентов системы алгебраичных линейных уравнений. При этом учитываем начальные условия:

$$S_w(x,0) = S_w^0(x), \qquad S_o(x,0) = S_o^0(x),$$
$$p_o(x,0) = p_o^0(x)$$

и граничные условия:

$$p_o(0,t) = p^{inj}, \quad p_o(1,t) = p^{prod},$$
$$S_w(0,t) = {S_w}^{inj}, \ \frac{\partial S_w}{\partial x}(1,t) = 0,$$

$$S_o(0,t) = 0, \qquad \frac{\partial S_o}{\partial x}(1,t) = 0$$

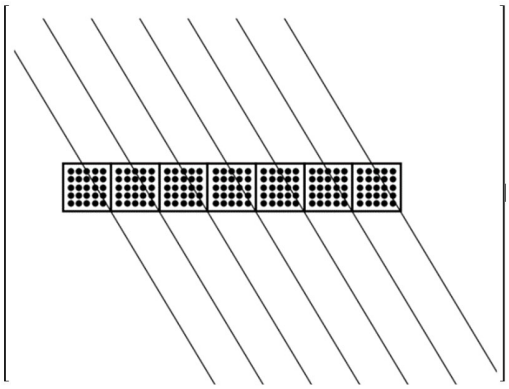


Рисунок 2 – Блочная матрица

Полученная система линейных уравнений (24) приводится к виду (1).

Решение системы линейных уравнений вида (1) является актуальной задачей в алгебре разреженных матриц. Здесь, A – разреженная матрица размера $n \times n$, которую можно получить написанием уравнений (24) для каждой точки разностной схемы. Для нашей задачи был выбран метод GMRES.

2.2 Метод обобщённых минимальных невязок (GMRES) в составе итерационного алгоритма

GMRES (Generalized minimal residual method) – метод обобщенных минимальных невязок, метод Крыловского типа для решения систем линейных уравнений общего вида. Данный метод является итерационным [17].

Рассматривается система линейных уравнений с разреженной невырожденной матрицей A размера $n \times n$ вида (1).

Рассмотрим подпространство Крылова K_m:

$$K_m = K_m(A, b) = span\{b, Ab, A^2b, \dots, A^{m-1}b\}$$

Можем представить любой вектор x из подпространства $x_0 + K_m$ в форме

$$x_m = x_0 + V_m y, \tag{25}$$

где V_m – матрица, составленная из векторов ортонормированного базиса подпространства Крылова K_m, а y – вектор размерности m. В таком случае невязка уравнения может определяться как функция вектора y

$$J(y) = \|b - Ax\| = \|b - A(x_0 + V_m y)\| \tag{26}$$

Из-за того, что столбцы матрицы V_{m+1} составляют ортонормированную систему векторов, выполняется следующее соотношение:

$$J(y) = \|b - A(x_0 + V_m y)\| = \|\beta e_1 - \bar{H}_m y\| \quad (27)$$

Метод обобщенных минимальных невязок аппроксимирует точное решение системы (1) вектором x_m, взятым из подпространства $x_0 + K_m$, который служит для минимизации невязки (26). Вектор x_m может быть получен как

$$x_m = x_0 + V_m y_m, \quad (28)$$

где V_m – матрица, составленная из векторов ортонормированного базиса подпространства Крылова K_m. Вектор y_m может находиться как решение линейной задачи наименьших квадратов размера $(m+1) \times m$, где $m \ll n$.

$$y_m = \arg\min_y \|\beta e_1 - \bar{H}_m y\|. \quad (29)$$

Теперь сформулируем алгоритм метода обобщенных минимальных невязок. Его входные параметры – выбранное число векторов подпространства Крылова $m \ll n$ и начальное приближение x_0.

1. Предварительный шаг: вычислить $r_0 = b - Ax_0$ и $\beta = r_0/\beta$.
2. Задать нулевую матрицу $\bar{H}_m$ размера $(m+1) \times m$.
3. Для $j = 1,2,3, \dots, m$ выполнить
    a. Вычислить $w_j = Av_j$
    b. Для $i = 1,2,3, \dots, j$ вычислить $h_{ij} = (w_j, v_j), w_j = w_j - h_{ij} v_i$
    c. Вычислить $h_{j+1,j} = \|w_j\|$
    d. Если $h_{j+1,j} = 0$, то $m = j$ и перейти на шаг 3
    e. Вычислить $v_{j+1} = w_j / h_{j+1,j}$
4. Вычислить $y_m = \arg\min_y \|\beta e_1 - \bar{H}_m y\|$ и $x_m = x_0 + V_m y_m$.

Вычисление y_m предполагает решение системы линейных уравнений

$$\bar{H}_m y = \beta e_1, \quad (30)$$

Видно что данная система является переопределенной (число строк больше числа столбцов) и поэтому должна решаться в смысле наименьших квадратов. Для решения этой задачи матрицу приводят к верхнетреугольному виду с помощью вращений Гивенса. На шаге i матрица вращения Ω_i имеет размеры $(m+1) \times (m+1)$ и записывается в следующем виде:

$$\Omega_i = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c_i & s_i & & \\ & & & -s_i & c_i & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}$$

где $c_i^2 + s_i^2 = 1$. Коэффициенты c_i, s_i выбираются так, чтобы обнулить коэффициент $h_{i+1,i}$. Таким образом, для i-й строки получим:

$$s_i = \frac{h_{i+1,i}}{\sqrt{h_{i,i}^2 + h_{i+1,i}^2}}, c_i = \frac{h_{i,i}}{\sqrt{h_{i,i}^2 + h_{i+1,i}^2}} \quad (31)$$

Приведение системы (24) к верхнетреугольному виду сводится к последовательному домножению обеих частей системы на матрицы поворотов $\Omega_1, \Omega_2, \ldots, \Omega_m$. После чего решается треугольная система, полученная после удаления последней строки из матрицы $\overline{H}_m^{(m)} = \Omega_1 \Omega_2 \ldots \Omega_m \overline{H}_m$ и последнего элемента правой части $\overline{g}_m = \Omega_1 \Omega_2 \ldots \Omega_m g_m$. При этом имеется возможность показать, что для решения y_m невязка $\|\beta e_1 - \overline{H}_m y\|$ совпадает с последним элементом γ_{m+1} вектора правой части $\overline{g}_m$. Данное значение совпадает с невязкой исходной системы уравнений, т.е.

$$\|r_m\| = \|b - Ax_m\| = |\gamma_{m+1}| \quad (32)$$

Это значение может использоваться в критерий остановки метода. При выполнении вращений Гивенса на i-м шаге вычисляется элемент γ_{i+1}, следуя из этого, можно проверить критерий невязки $\|r_i\| < \varepsilon$ или относительной невязки $\|r_i\|/\|b\| < \varepsilon$[1].

2.3 Предобуславливание и его влияние на сходимость

Одним из важных свойств итерационных методов является сходимость, количество итераций необходимых для достижения условия сходимости. Для сокращения числа итераций итерационных методов применяется предобуславливание – явное либо неявное преобразование исходной системы линейных уравнений, направленное на упрощение её решения. Современные подходы к предобуславливанию включают методы аппроксимации матрицы системы (ILU, IQR, ILQ) и методы аппроксимации её обратной матрицы: полиномиальные, разреженные (AINV) и основанные на аппроксимации

факторизованной формы (FSAI, SPAI и др.). Наиболее распространённым является неполное LU-разложение (ILU) матрицы A [23].

Рассмотрим предобуславливание с помощью домножения исходной системы на некоторую матрицу P^{-1} , т.е. переход к системе вида

$$P^{-1}Ax = P^{-1}b \tag{33}$$

Матрица P называется предобуславливателем.

Одним из широко распространенных методов для нахождения предобуславливателя является неполное LU-разложение (ILU-факторизация) исходной матрицы A. LU-разложение является модификацией метода Гаусса.

Матрицы L и U можно найти следующим образом:

1. $u_{1j} = a_{1j},\ j = 1 \dots n$
2. $l_{j1} = \frac{a_{j1}}{u_{11}},\ j = 2 \dots n (u_{11} \neq 0)$

для $i = 2 \dots n$

1. $u_{ij} = a_{ij} - \sum_{k=1}^{i} l_{ik}u_{kj},\ j = i \dots n$
2. $l_{ji} = \frac{1}{u_{ii}}\left(a_{ji} - \sum_{k=1}^{i} l_{jk}u_{ki}\right),\ j = i+1 \dots n$

где a_{ij}, l_{ij}, u_{ij} – элементы с индексами i, j матриц A,L и U соответственно.

Неполное разложение предполагает представление матрицы A в форме

$$A = LU + R, \tag{34}$$

где матрицы L и U являются нижне и верхнетреугольными матрицами соответсвенно, с портретами, совпадающими с нижним и верхним треугольниками матрицы A, а матрица R невязка разложения. После проведения ILU-факторизации в качестве предобуславливателя можно использовать матрицу:

$$P = LU \tag{35}$$

Алгоритм Gmres с предобуславливателем ILU(0) будет выглядеть следующим образом [57]:

1. Предварительный шаг: вычислить $Pr_0 = b - Ax_0$ и $\beta = r_0/\beta$.
2. Задать нулевую матрицу $\bar{H}_m$ размера $(m+1) \times m$.
3. Для $j = 1,2,3, \dots, m$ выполнить
   a. Вычислить $Pw_j = Av_j$
   b. Для $i = 1,2,3, \dots, j$ вычислить $h_{ij} = \left(w_j, v_j\right), w_j = w_j - h_{ij}v_i$
   c. Вычислить $h_{j+1,j} = \left\|w_j\right\|$
   d. Если $h_{j+1,j} = 0$, то $m = j$ и перейти на шаг 3
   e. Вычислить $v_{j+1} = w_j/h_{j+1,j}$

4. Вычислить $y_m = \arg \min_y \|\beta e_1 - \bar{H}_m y\|$ и $x_m = x_0 + V_m y_m$.

Хотя изменения кажутся незначительными, стоит вопрос как можно помножить систему на предобуславливатель. Например, на шаге 3.а) $Pw_j = Av_j$, можно перенести предобуславливатель на правую сторону: $w_j = P^{-1}Av_j$. Но данное изменение приведет к нахождению обратной матрицы P^{-1} разреженной матрицы P, что последует за собой переполнение матрицы. Выбранный нами подход представляет из себя решение системы:

$$LUw_j = Av_j \tag{36}$$

где матрицы L и U являются нижне и верхнетреугольными матрицами соответсвенно как было сказано ранее. Решение этой системы можно разделить на два этапа:

1. Решение системы $Lx = Av_j$
2. Решение системы $Uw_j = x$

Так как в обеих случаях матрицы являются треугольными, решение этих систем приводится к обратному ходу метода Гаусса.

Такая же схема используется на шаге 1.

На рисунке 3 ниже можно увидеть общую схему работы алгоритма, диаграмму активностей. В схеме показана работа алгоритма от выбора начального приближения x_0 и до получения приближенного решения x_m.

На рисунке 4 показана диаграмма активностей метода Newton-ILU(0)-GMRES. В схеме показана работа метода с начальной инициализацией до окончания моделирования. Также можно увидеть каким образом выполняются итерации метода Ньютона, и что внутри каждой итерации Ньютона решается СЛАУ методом ILU(0)-GMRES. Разветвление показывает что метод учитывает переход состоянии внутри пласта от ненасыщенного к насыщенному, и обратно.

Также нужно отметить, что, как было сказано ранее, матрица A из уравнения (20) является разреженной. Разреженными матрицами называются такие матрицы, в которых количество ненулевых элементов значительно меньше общего числа элементов [58]. Хранение таких матриц в памяти в традиционном виде, то есть полное хранение всех элементов является не оптимальным, так как нулевые элементы будут занимать много ненужного объема памяти а также будут затормаживать доступ к ненулевым элементам. В данной работе для хранения разреженных матриц был использован формат CSR.

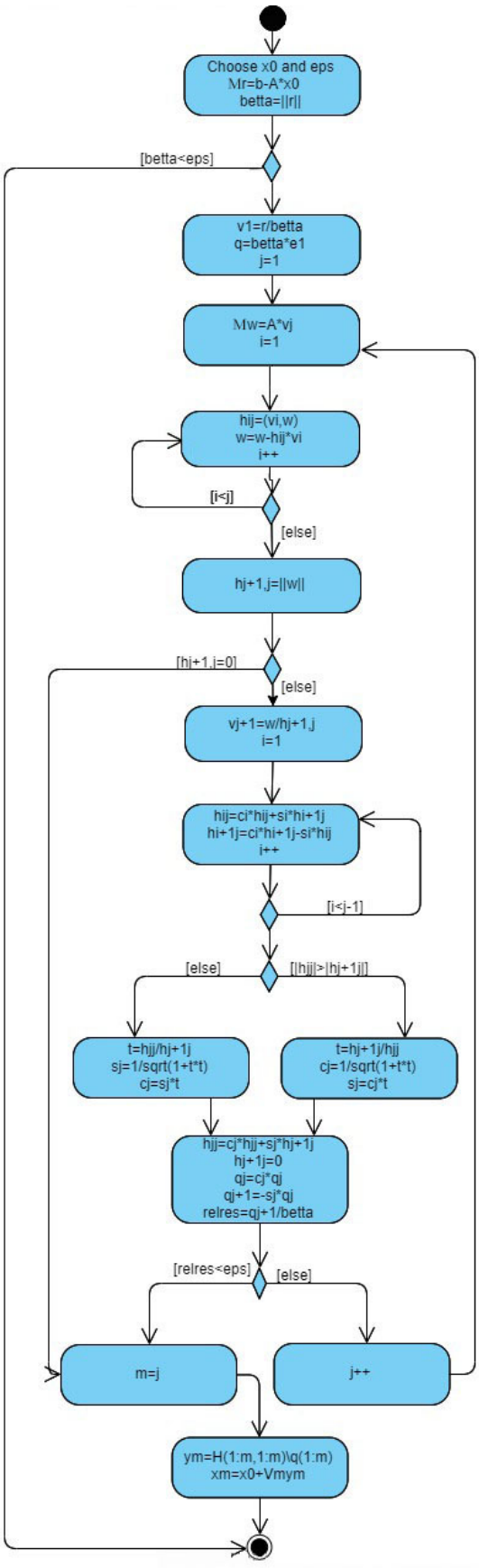


Рисунок 3 – Схема выполнения алгоритма ILU(0)-GMRES

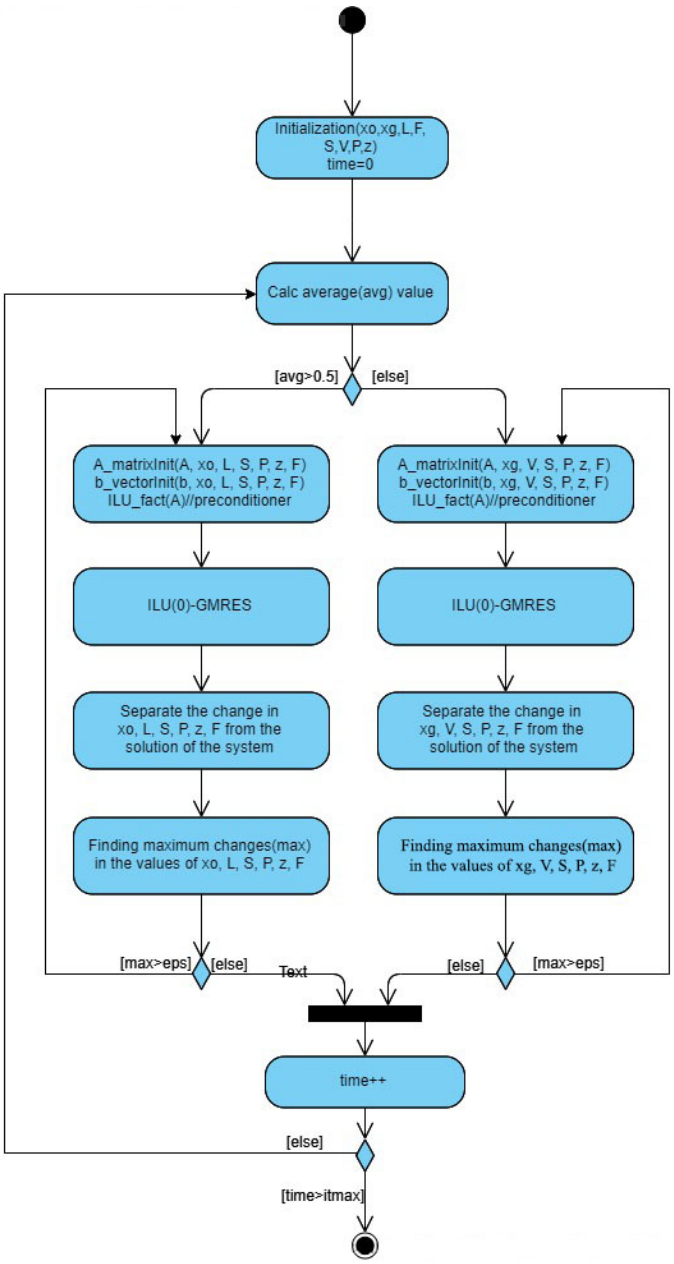


Рисунок 4 – Схема выполнения алгоритма Newton-ILU(0)-GMRES

2.4 Сходимость метода

Разработанный алгоритм численного решения задачи неравновесной фильтрации использует метод Ньютона для линеаризации системы нелинейных алгебраических уравнений, возникающей после пространственно-временной дискретизации исходной модели. На каждом шаге метода Ньютона система линейных уравнений решается итерационным методом GMRES в сочетании с предобуславливателем ILU(0).

Метод Ньютона обладает локальной квадратичной сходимостью при условии, что начальное приближение находится в окрестности искомого решения, функция правой части системы непрерывна и дифференцируема, а матрица Якоби является невырожденной. В рамках построенного алгоритма в

качестве начального приближения на текущем временном шаге используются значения, полученные на предыдущем слое, что обеспечивает практическое выполнение условий сходимости и стабильную работу итерационного процесса.

Метод GMRES, применяемый для решения линеаризованных систем, обеспечивает сходимость при наличии эффективного предобуславливателя. Используемое в работе предобуславливание ILU(0) позволяет существенно ускорить сходимость метода, особенно при решении систем с несимметричными и недиагонально доминирующими матрицами. Кроме того, применение предобуславливания способствует более компактному распределению собственных значений результирующей матрицы, что положительно влияет на сходимость итерационного метода.

Также при решении линеаризованных систем методом GMRES устойчивость оценивается по норме невязки:

$$\|r_m\| = \|b - Ax_m\| < \varepsilon$$

что контролирует точность приближённого решения. Применение предобуславливания улучшает спектральные свойства матрицы A, снижая её условность в выбранной норме и обеспечивая устойчивое поведение метода.

Совместная сходимость итерационного метода Ньютона и вложенного метода GMRES обеспечивается за счёт согласованного управления точностью: на ранних итерациях допускается более грубое приближение в решении линейной задачи, в то время как на финальных итерациях достигается заданная точность. Такой подход предотвращает накопление ошибок и способствует быстрой сходимости всей процедуры.

С точки зрения устойчивости, применение неявной схемы по времени обеспечивает устойчивость численного метода, в том числе при достаточно крупных шагах по времени. Дополнительную устойчивость обеспечивают структура метода Ньютона, предобуславливание GMRES, а также свойства используемой модели.

3 РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ/РАСПРЕДЕЛЕННЫХ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

Параллельные алгоритмы представляют собой подход к решению вычислительно сложных задач, при котором обработка данных выполняется одновременно на нескольких вычислительных узлах или процессорах. Главная цель разработки параллельных алгоритмов заключается в увеличении производительности и сокращении времени вычислений за счёт эффективного использования ресурсов многопроцессорных систем и кластеров.

Одной из наиболее распространённых техник при разработке параллельных алгоритмов является декомпозиция данных. Данный подход подразумевает разделение набора данных на части, которые могут быть обработаны параллельно.

В зависимости от характера задачи выделяют следующие виды декомпозиции данных:

- Пространственная декомпозиция: применяется для задач, в которых данные организованы в виде сетки или области, например, при моделировании физических процессов в пространстве. Каждый процесс обрабатывает свою часть данных, связанную с определённой областью пространства.
- Декомпозиция по данным: используется, если данные представляют собой большие массивы или списки. Каждый процесс обрабатывает определённый фрагмент массива, что снижает общую сложность и повышает производительность.
- Функциональная декомпозиция: подразумевает разделение задачи по функциональным модулям. Каждый процесс отвечает за выполнение определённой функции или этапа вычислений.

Декомпозиция данных позволяет значительно уменьшить время выполнения задачи за счёт равномерного распределения нагрузки и эффективного использования ресурсов системы. Важно отметить, что при разработке параллельных алгоритмов необходимо учитывать взаимозависимость данных между различными частями, чтобы минимизировать накладные расходы на межпроцессорные коммуникации и синхронизацию.

В данной работе рассматривается подход, основанный на декомпозиции по данным, при котором вычислительная нагрузка распределяется путём разбиения матриц и векторов при решении систем линейных уравнений. Данный метод позволяет равномерно распределить данные между процессами и обеспечить эффективное использование вычислительных ресурсов.

Основные аспекты разработки алгоритмов с декомпозицией по данным включают:

- Разделение данных: матрицы и векторы разбиваются на подмножества, каждая из которых назначается отдельному процессу.
- Распределённые вычисления: каждый процесс выполняет операции над своей частью данных, минимизируя необходимость обмена данными.

- Коммуникация между процессами: обеспечивается эффективный обмен граничными данными между процессами для корректного выполнения вычислений.
- Балансировка нагрузки: равномерное распределение данных по процессам с учётом их вычислительных возможностей, чтобы избежать узких мест и простаивания ресурсов.

Использование декомпозиции по данным особенно эффективно при решении крупных задач линейной алгебры, таких как методы решения разреженных линейных систем, итерационные методы Крылова и факторизация разреженных матриц.

3.1 Построение параллельного алгоритма метода GMRES на основе MPI

Суть параллельной реализации заключается в адаптации алгоритма GMRES для параллельного выполнения. Поскольку наиболее вычислительно затратные задачи при решении композиционной модели приходятся на метод GMRES, оптимизация этого этапа является ключевой. Параллелизация алгоритма GMRES позволяет распределить матричные операции, из которых состоит данный метод, между несколькими процессорами [59]. Это распределение значительно ускоряет вычисления за счет использования возможностей одновременного выполнения современных многоядерных и распределенных вычислительных сред.

Включение этого параллельного алгоритма GMRES в процесс решения не только ускоряет вычисления, но и делает подход масштабируемым, что критически важно для решения масштабных задач, присущих композиционному моделированию. К тому же, рассматриваемый подход требует много ресурсов памяти, особенно в трехмерном случае, которые могут не помещаться в памяти одного вычислительного узла. Данная особенность является еще одной причиной потребности в распределенных вычислениях.

Алгоритм GMRES содержит много матрично-векторных операций, которые могут быть распределены между процессорами. Распределение таких операций можно увидеть на примере нахождения приближенного решения $x_m = x_0 + V_m y_m$, где происходит умножение матрицы V_m на вектор y_m и сложение полученного вектора и вектора x_0. Декомпозицию можно увидеть на рисунке 5.

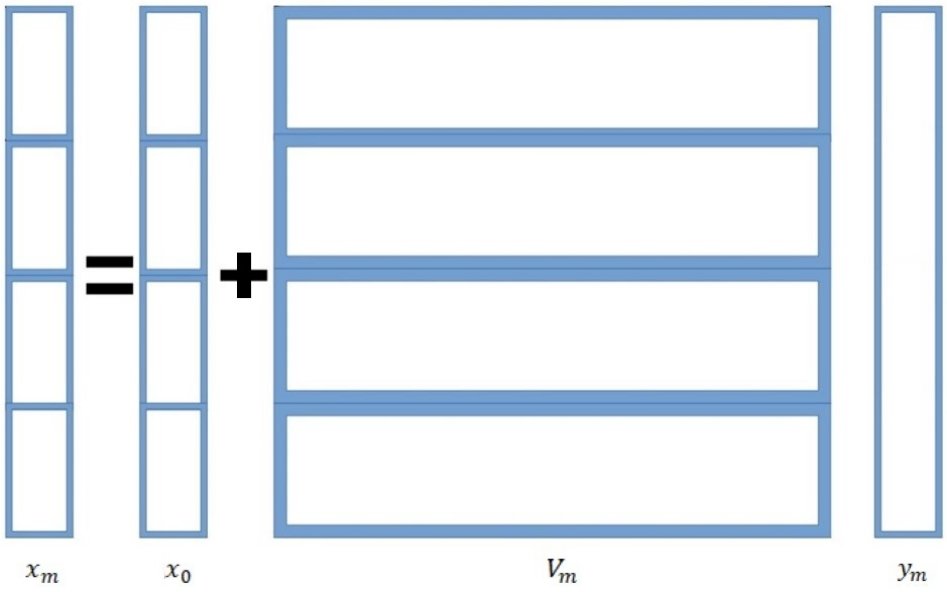


Рисунок 5 – Декомпозиция данных

Из этого рисунка видно что плотная матрица V_m разделятся по строкам, вектор x_0 разделяется поэлементно, а вектор y_m рассылается полностью. Каждый процесс вычисляет элементы x_m соответствующие полученным частям входных данных.

Другой пример, вычисление невязки $r_0 = b - Ax_0$ (рис. 6). Главное отличие этого примера от предыдущего в том что здесь матрица и его разделение является блочным.

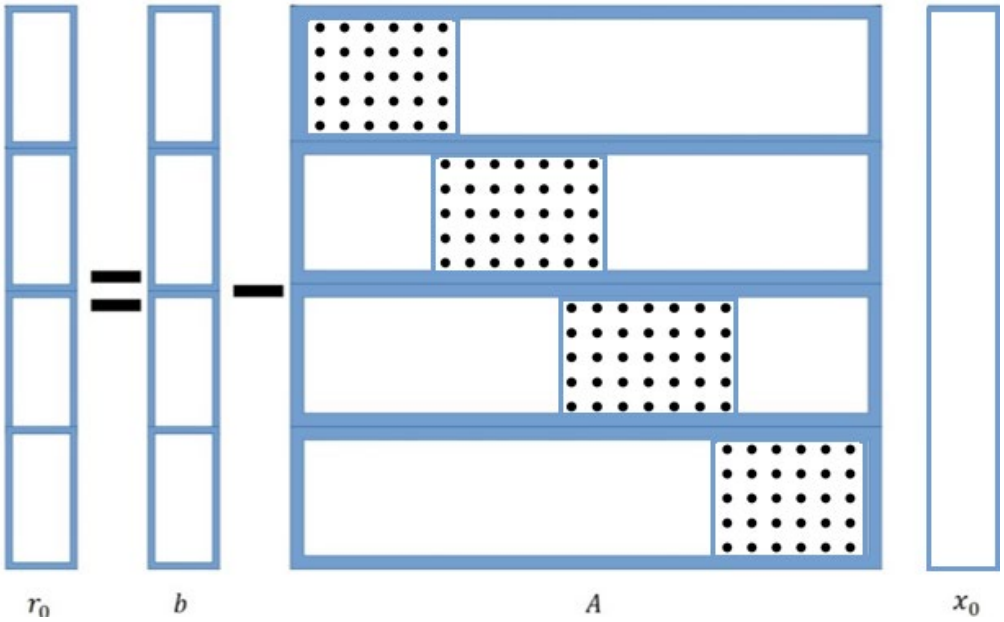


Рисунок 6 – Декомпозиция данных при вычислении невязки

При этом нужно учесть что матрица в памяти представляет из себя три одномерных массива, как было сказано ранее. Данный рисунок показывает как

может выглядеть разделение матрица A, если восстановить его в полный вид. Для всех матрично-векторных операций была применена декомпозиция подобного вида.

Остановимся отдельно на умножении на предобуславливатель: $LUw_j = Av_j$. Как было отмечено в подглаве 2.3, данное умножение происходит в два этапа по схеме обратного хода метода Гаусса. Данные операции не могут быть выполнены простым разделением данных по процессам, так как имеется явная зависимость данных.

Подробно распишем решение системы:

$$Lx = Av_j. \tag{37}$$

здесь Av_j представляет собой вектор, поэтому для удобства запишем в следующем виде:

$$Lx = y \tag{38}$$

В покомпонентной форме, для трех неизвестных данная система выглядит таким образом:

$$\begin{bmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

Нахождение первого элемента вектора x не заставит проблем:

$$x_0 = \frac{y_0}{L_{0,0}} \tag{39}$$

Далее, зная значение x_0, можно найти следующий элемент:

$$x_1 = \frac{y_1 - x_0 L_{1,0}}{L_{1,1}} \tag{40}$$

Формула в общем виде:

$$x_i = \frac{y_i - \sum_{j=0,\dots,i-1} x_j L_{i,j}}{L_{i,i}}, \text{для } i=0,\dots,n-1 \tag{41}$$

При построчном разделении данных, в каждой итерации по *i,* параллельным процессам нужно будет знать значение с предыдущих итераций, что означает что процессы будут выполнять вычисления последовательно. Поэтому нами был разработан параллельный конвейерный алгоритм умножения на предобуславливатель. Суть метода в том что каждый процесс после вычисления одного элемента вектора неизвестных x_i отправляет этот элемент

остальным процессам, которые используя этот элемент производят параллельно частичное суммирование $\sum_{j=0,\dots,i-1} x_j L_{i,j}$ из формулы (41). На рисунке ниже показана схема конвейерного алгоритма на примере решения системы размера 6x6. Разными цветами указано разделение даных по процесам.

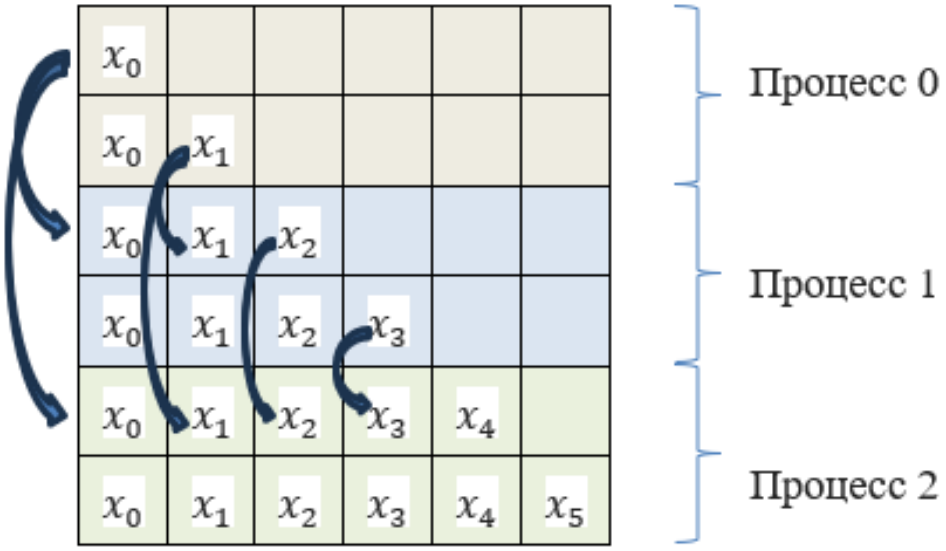


Рисунок 7 – Декомпозиция данных и конвейерная схема передачи

На рисунке указаны на какие элементы матрицы L умножается вектор x. Первым вычисляется значение x_0 процессом 0 и отправляет это значение следующим процессам, которые производят частичное суммирование. На следующем шаге процесс 0 вычисляет значение x_1 и также отправляет следующим процессам. Далее процесс 1 легко вычисляет x_2 так как сумма $\sum_{j=0,\dots,i-1} x_j L_{i,j}$ уже вычислена. Такой же процесс повторяется при масштабировании алгоритма на системы большего размера и на большее количество процессов.

Недостатком этой схемы является простаивание процессов после нахождения своей части вектора неизвестных. Например, после второго шага алгоритма по схеме, показанной на рисунке 6, процесс 0 простаивает на остальных четырех шагах, пока работаю процессы 1 и 2. Решить данную проблему помогло разделение по схеме циклического сдвига.

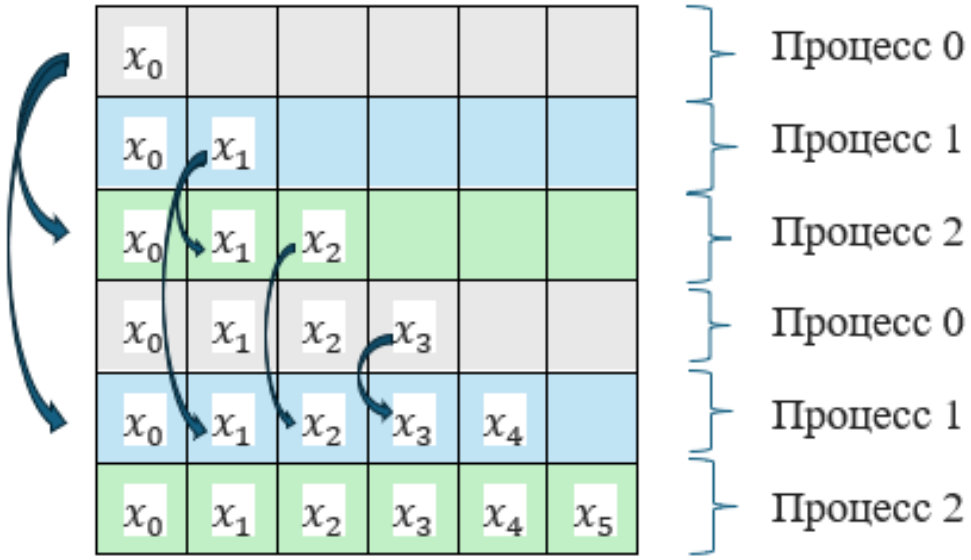


Рисунок 8 – Декомпозиция данных и конвейерная схема передачи при цикличном сдвиге

Цикличное разделение данных можно увидеть из цветов строк. Если при обычном делении данных для этого примера нулевой процесс простаивал 4 шага алгоритма, при цикличном делении нулевой процесс простаивает 2 шага, поэтому данный вариант частично решает проблему. Но при большом количестве данных разница между двумя вариантами декомпозиции становится существенной.

Для реализации параллельного алгоритма был использован стандарт MPI. Все коммуникаций были выполнены с помощью стандартных функций MPI. Данная технология сильно зависит от квалификации системного программиста. Например, описанная декомпозиция является строгим и не может быть скорректирована во время выполнения программы, что может привести к простаиванию вычислительных узлов при некорректном распределении работ. Это приводит к такой дополнительной проблеме, что специалист, занимающийся численными решениями, должен обладать дополнительными знаниями и в системном программировании. Решением может быть или использование готовых библиотек и пакетов в которых нужные методы уже были реализованы, или системы с автоматизацией конструирования параллельных программ. Автоматизация здесь означает автоматический перенос последовательных программ или программ написанных на высокоуровневом языке на программы запускаемые в параллельных системах. Система LuNA относится к таковому решению.

3.2 Разработка фрагментированного алгоритма численного решения

в подглаве 3.1. При разделении на фрагменты сохраняется данная схема. Например, при разделении вектора из

n элементов на fg_count фрагменты, размер каждого фрагмента будет составлять n/fg_count [60], [61].

Как было сказано ранее, матрица A из уравнения (1) является разреженной. Разреженными матрицами называются такие матрицы, в которых количество ненулевых элементов значительно меньше общего числа элементов [58]. Хранение таких матриц в памяти в традиционном виде, то есть полное хранение всех элементов является не оптимальным, так как нулевые элементы будут занимать много ненужного объема памяти а также будут затормаживать доступ к ненулевым элементам.

Хранение разреженной матрицы в памяти должно обеспечивать:

1. Экономию памяти
    - N×M – размер матрицы
    - NZ – количество ненулевых элементов
    - NZ ≪ N×M
2. Быстрый доступ к нулевым и ненулевым элементам по индексу

В данной работе для хранения разреженных матриц был использован формат CSR.

Формат сжатых строк (Compressed Row Storage, CRS, или Compressed Sparse Rows, CSR) – это формат хранения разреженных матриц построчно.

Структура хранения:

Матрица хранится в памяти в виде трёх одномерных массивов следующего вида:

1. A – одномерный массив, где, начиная с первой и до последней строки, записываются все ненулевые элементы a_{ij} построчно, а нулевые элементы пропускаются.
2. LJ – одномерный массив, содержащий номер столбца каждого ненулевого элемента.
3. LI – одномерный массив, в котором содержится индекс первого ненулевого элемента каждой строки в массиве A.

Элементы i-й строки матрицы A находятся в диапазоне индексов от LI[i] до LI[i+1] - 1.

- Если строка i состоит только из нулей, т.е. строка пустая, то LI[i] = LI[i+1].
- Если матрица A имеет n строк, то длина массива индексов LI равна n+1.

При таком методе хранения затраты памяти для хранения разреженной матрицы размером n×m с NZ ненулевыми элементами составляют:

- NZ ячеек для массива элементов (A)
- NZ ячеек для массива номеров столбцов (LJ)
- n + 1 ячеек для массива индексов строк (LI)

Этот метод представления также является полным и отсортированным, поскольку элементы в каждой строке упорядочены в порядке возрастания индексов столбцов.

В таком же формате хранятся матрицы L и U, получаемые в результате ILU-факторизации.

В фрагментированном варианте изначально три массива представляются в виде трех фрагментов A[0], A[1], A[2]. Нужно отметить, что индексация на языке LuNA отличается от индексации на C++. Здесь индексы больше означают название, чем их порядковый номер или шаг после начала нулевого элемента. При разделении каждого массива на ФД в количестве $FG_COUNT, добавляется дополнительный индекс. Данная схема показана в рисунке ниже.

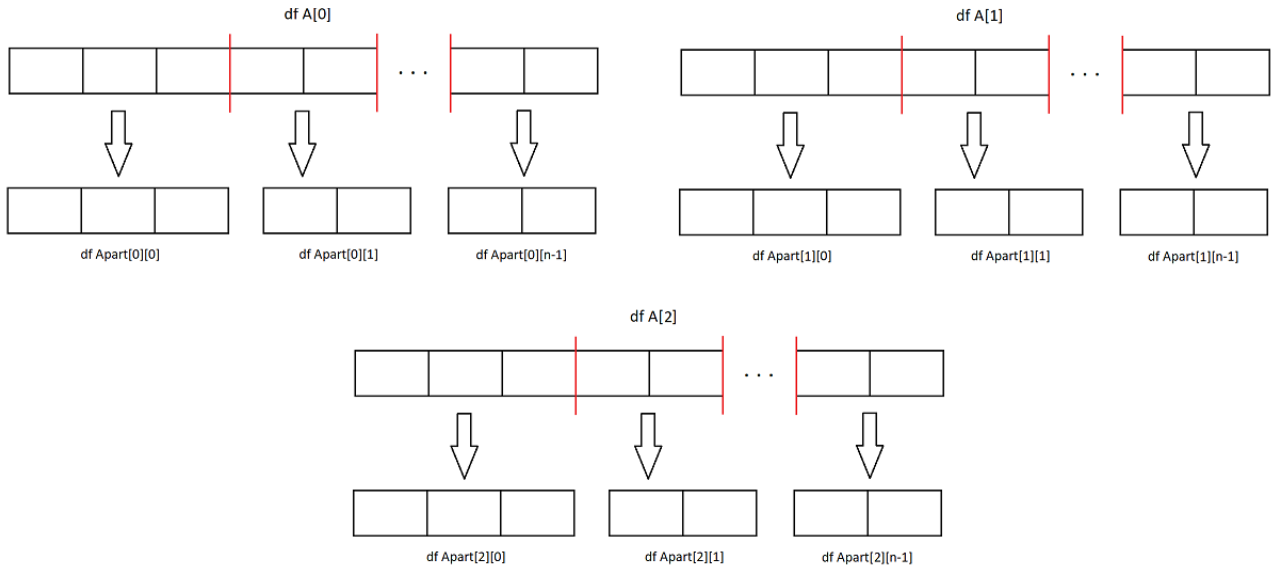


Рисунок 9 – Разделение матрицы A на фрагменты

Данная схема также актуальна для разделения матрицы L и U.

Прежде чем перейти к фрагментированному алгоритму, представим алгоритм решения СЛАУ в виде псевдокода:

Алгоритм 1:

1. $r_0 = P^{-1}(\mathrm{b} - Ax_0)$
2. $\beta = \|r_0\|$
3. $v_1 = r_0/\beta$
4. for i=1,..,m
5. $w = P^{-1}Av_i$
6. for k=1,..,i
7. $h_{k,i} = (w, v_k)$
8. $w = w - h_{k,i}v_k$
9. end k
10. $h_{i+1,i} = \|w_i\|$
11. $v_{k+1} = \frac{w}{h_{i+1,i}}$
12. end i
13. $y_m = \arg\min_y \|\beta e_1 - \bar{H}_m y\|$
14. $x_m = x_0 + V_m y_m$

Данное представление повторяет алгоритм из подглавы 2.3 в виде, близком к коду.

Порождение фрагментов не контролируется пользователем. Каждый ФВ начинает выполнение при готовности всех входных ФД. На рисунке ниже можно увидеть примерную схему нахождения невязки с Алгоритма 1, строка 1.

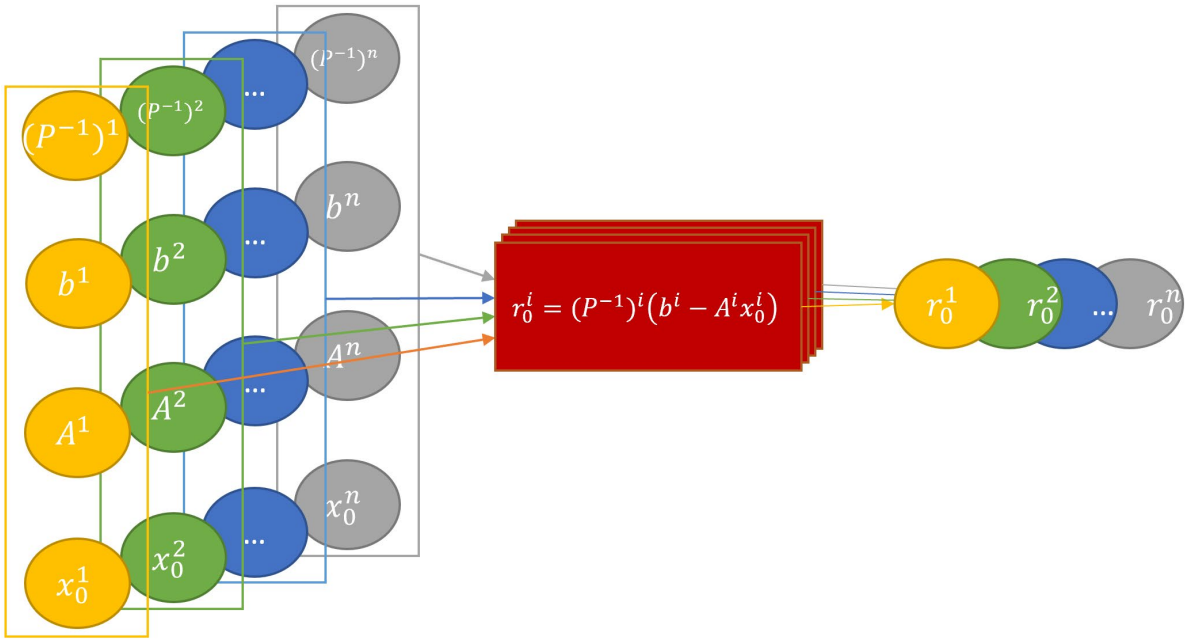
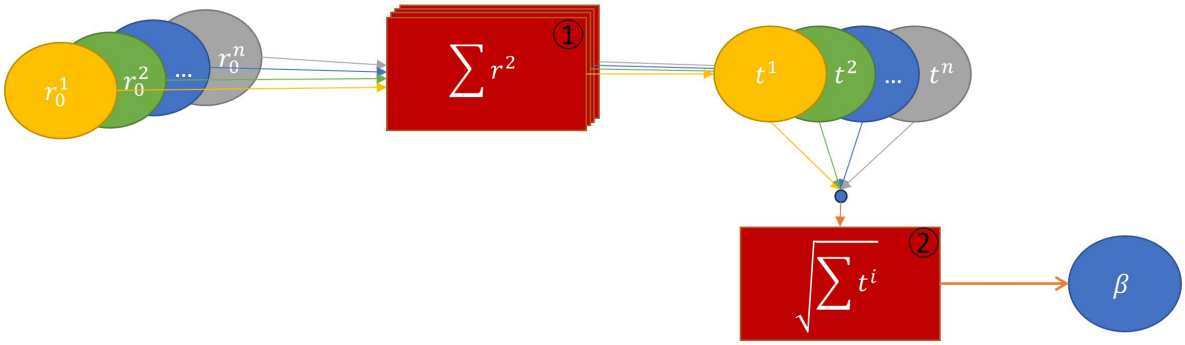


Рисунок 10 – Схема фрагментированного нахождения невязки

Видно, что матрицы и вектора разделяются на n фрагментов. Разными цветами показаны наборы ФД, которые являются входными для различных ФВ со схожими операциями (красный прямоугольник). Например, при готовности всех ФД желтого цвета, начинается выполнение соответствующего ФВ и порождается выходной ФД, отмеченный желтым. Следующий пример, нахождение нормы вектора $\beta = \|r_0\|$, строка 2 Алгоритма 1.



11 – Схема фрагментированного нахождения нормы вектора

Нахождение нормы разделено в два шага:

- первый шаг: нахождение локальных сумм;

- второй шаг: нахождение нормы – квадратного корня от глобальной суммы.

На рисунке можно увидеть нумерацию на ФВ, которые означают эти два шага. При этом выполняется множество ФВ под номером 1, с общей структурой, для отдельных ФД. Например, отдельный ФВ порождает ФД t^1, временный фрагмент для локальной суммы, при готовности ФД r_0^1, первой части невязки на нулевой итерации. Но второй ФВ начинает выполнение только при готовности всех временных локальных сумм и порождает один единственный ФД.

Далее представим строки 6-9 Алгоритма 1 на языке LuNA:

Листинг 1:

```
1.  for k=0..j
2.  {
3.      for i=0..$FG_COUNT-1
4.      {
5.          multipleVtow(i,NX[i],Vpart[itr][k][i],wpart[itr][j][k+1][i],Htemp[itr][j][k][i]);
6.      }
7.      normsum(Htemp[itr][j][k][0],Htemp[itr][j][k][1],spart[itr][j][k][0]);
8.      for i=1..$FG_COUNT-2
9.      {
10.         normsum(spart[itr][j][k][i-1],Htemp[itr][j][k][i+1],spart[itr][j][k][i]);
11.     }
12.     executeHi(k,j,m[0],spart[itr][j][k][$FG_COUNT-2],H[itr][j][k],H[itr][j][k+1]);
13.     for i=0..$FG_COUNT-1
14.     {
15.         execw(j, k, i, m[i], NX[i], wpart[itr][j][k+1][i], H[itr][j][k+1], Vpart[itr][k][i],
            wpart[itr][j][k+2][i]);
16.     }
17. }
```

Здесь функции multipleVtow, normsum, executeHi и execw импортируются из отдельного файла и описываются там же на языке высокого уровня. Строки с 3 по 12 из Листинга 1 соответствуют реализации скалярного умножения $h_{k,i} = (w, v_k)$. Этот процесс можно разделить на два шага, сначала отдельные ФВ с помощью функции multipleVtow находим суммы поэлементного умножения ФД Vpart и wpart и записываем в ФД Htemp. Затем идет суммирование этих временных ФД функцией normsum и запись результата в ФД H функцией executeHi. Далее происходит обновление $w = w - h_{k,i}v_k$ функцией execw.

На рисунке 12 представлена схема фрагментированного алгоритма для вычисления выражения $h_{k,i} = (w, v_k)$, представляющего собой скалярное произведение векторов w и v_k.

Векторы w и v_k разделяются на n фрагментов данных и обозначаются как w_p[i][j] и v_p[i][j], где индекс j=1…n соответствует номеру фрагмента, а индекс i – номеру итерации метода Арнольди. Индексирование по i обеспечивает иммутабельность: на каждой итерации формируются новые фрагменты данных, а предыдущие могут быть удалены. При этом данные индексы не подразумевают наличие массивов или матриц в памяти – они служат лишь для уникальной идентификации фрагментов. На рисунке 12 рассмотрен пример, в котором

количество фрагментов равно трём. Соответствующие фрагменты вычислений (cf) параллельно определяют скалярные произведения частей данных, после чего все локальные суммы объединяются в единый фрагмент данных.

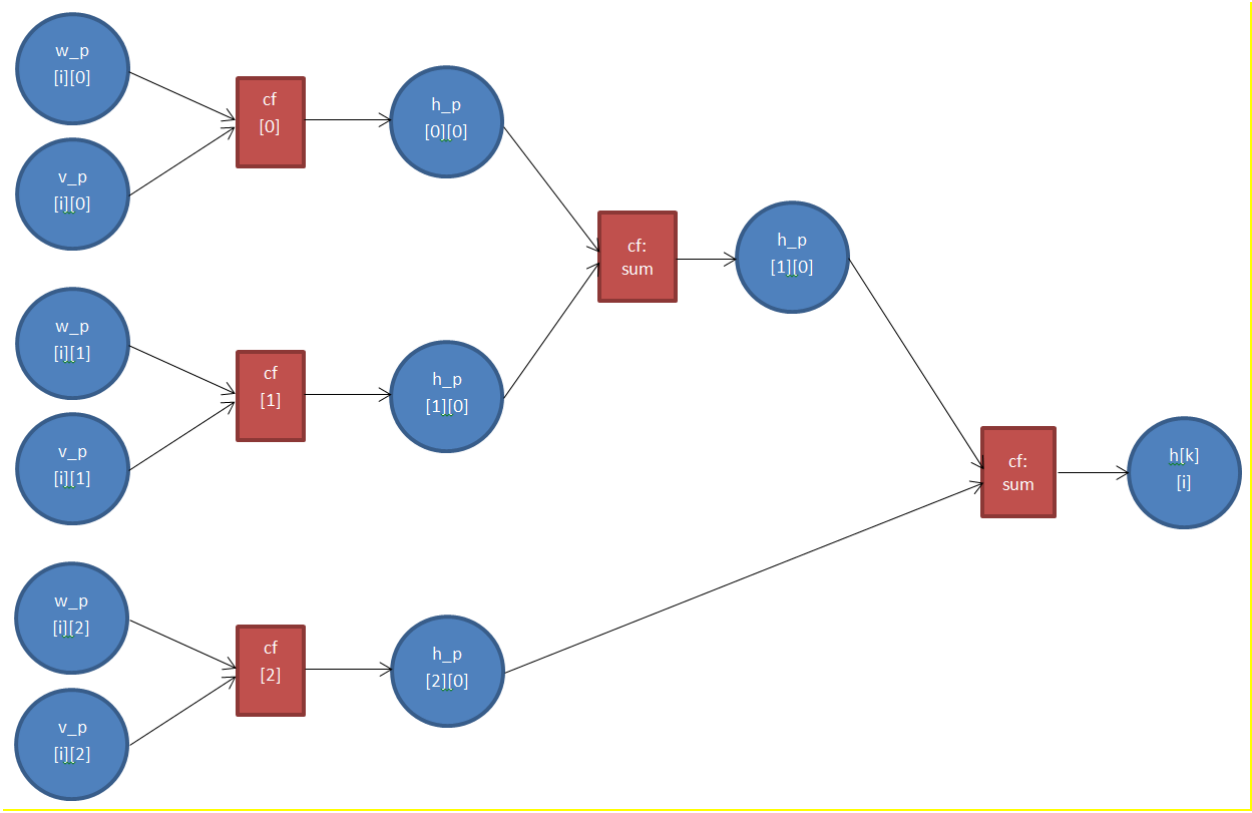


Рисунок 12 – Фрагментированный алгоритм вычисления коэффициента ортогонализации

В Листинге 1 представлены индексы, сопровождающие фрагменты данных. Эти индексы используются для различения фрагментов, создаваемых на разных этапах вычислений. Поскольку в системе LuNA фрагменты данных (ФД) являются иммутабельными, новые версии ФД порождаются с уникальными индексами.

Например, индекс [itr] обозначает итерацию метода Gmres: на каждой новой итерации создаются новые ФД с соответствующим индексом [itr]. При необходимости можно обращаться к предыдущим ФД, используя индекс [itr-1]. Аналогично работает индекс [k], который различает ФД, созданные на разных итерациях по переменной k.

Далее приведем общую схему фрагментированного алгоритма решения СЛАУ на рисунке 13. Кружочками показаны ФД, а прямоугольниками ФВ. На верхней правой стороне показана нумерация ФВ для удобного объяснения. Разделение на цвета позволит показать группировку некоторых ФД. Например, ФВ1, при готовности всех желтых входных ФД, начинает свою работу и порождает желтый выходной ФД.

Также после ФВ4 можно увидеть начало цикла строки 4 Алгоритма 1. А ФВ6 и ФВ7 соответствуют операциям цикла на строке 6 Алгоритма 1. ФВ с номера 5 по номер 9 показаны как множество, так как вычисляются множество раз внутри этих циклов.

По схеме может показаться что у ФВ4 и ФВ9 одни и те же выходные ФД. Эти ФД означают вектора ортонормированного базиса подпространства Крылова, и нижним индексом i показано что с каждым ФВ9 внутри цикла порождается новый вектор v. Также эти вектора соединяются в матрицу V_m и служат входным ФД для ФВ11. Таким же образом нижним индексом k показаны отдельные ФД вектора w для каждой итерации цикла. Выходные ФД ФВ6 и ФВ8 объединяются в матрицу $\bar{H}_m$ и являются входными ФД для ФВ10.

3.3 Сходимость фрагментированного алгоритма

Фрагментированный алгоритм численного решения задачи неравновесной фильтрации реализует ту же математическую схему, что и последовательная версия: для линеаризации системы нелинейных уравнений применяется метод Ньютона, а для решения получаемых систем линейных уравнений используется метод GMRES с предобуславливанием ILU(0). Ключевое отличие заключается в организации выполнения, основанной на декомпозиции данных и вычислений на отдельные фрагменты с их последующим независимым исполнением в системе LuNA.

Функциональная и дата-фрагментированная структура алгоритма не нарушает численную сходимость метода, так как сохраняется логика вычислений, порядок итераций, условия остановки и точность промежуточных решений. Согласованность между фрагментами обеспечивается за счёт системы зависимостей и механизмов синхронизации, встроенных в среду исполнения LuNA.

Численные эксперименты показали, что результаты, полученные фрагментированным алгоритмом, идентичны результатам последовательной реализации с точностью до машинного округления. Это подтверждает, что сходимость метода Ньютона и вложенного метода GMRES сохраняется при фрагментированном исполнении. Поскольку сами методы являются численно устойчивыми, при корректной организации фрагментов и соблюдении зависимостей их сходимость гарантирована и в параллельной среде.

Таким образом, фрагментированный алгоритм демонстрирует ту же численную сходимость, что и последовательный, обеспечивая при этом возможность масштабируемого и переносимого исполнения без потери математических свойств метода.

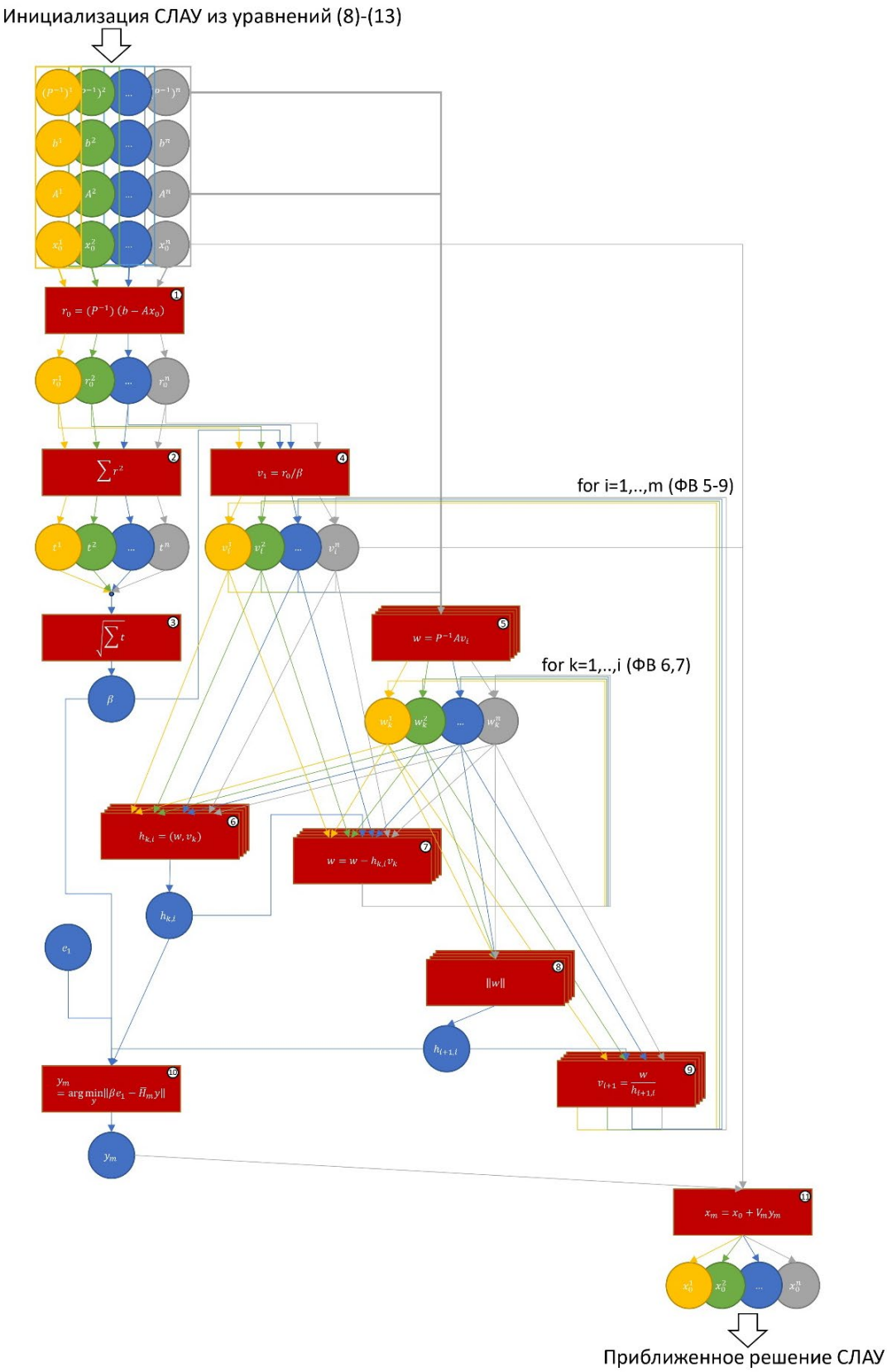


Рисунок 13 – Фрагментированный алгоритм решения СЛАУ методом ILU(0)-GMRES

3.4 Механизмы оптимизации

Для повышения эффективности фрагментированной реализации алгоритма численного решения задачи фильтрации в системе LuNA были использованы два механизма исполнения: уничтожающее потребление и воспроизведение трасс. Их применение позволяет сократить накладные расходы на управление памятью и устранить часть динамических решений в процессе исполнения, что может обеспечить прирост производительности.

3.4.1 Уничтожающее потребление

Система LuNA реализует принцип единственного присваивания фрагментам данных (ФД), при котором каждое изменение данных приводит к созданию нового ФД и удалению предыдущего [45]. Это может вызывать избыточные операции копирования памяти, особенно в итерационных фрагментах, где данные последовательно модифицируются.

В итерационных частях разработанного фрагментированного алгоритма активно обновляются векторы и промежуточные данные, связанные с решением системы линейных уравнений методом GMRES. В изначальной фрагментированной реализации каждое обновление таких данных приводило к созданию новых фрагментов данных, что вызывало дополнительные накладные расходы на копирование и размещение в памяти. Для устранения этой проблемы используется механизм уничтожающего потребления. В данной работе данный механизм был внедрён в тех участках фрагментированной программы, где ранее происходило копирование памяти. Вместо этого применялась функция grab() из системного интерфейса LuNA, позволяющая получить доступ к буферу существующего ФД и использовать его в качестве выходного без создания нового объекта. Такой подход позволяет перезаписывать данные на месте, снижая затраты на выделение памяти и улучшая локальность данных при исполнении. Этот подход оказался особенно полезен при построении и обновлении векторов подпространства Крылова и остатков, где множественные итерации приводят к повторяющемуся обновлению одних и тех же структур.

3.4.2 Воспроизведение трасс

Механизм воспроизведения трасс позволяет переисполнять фрагментированную программу в том же порядке, в котором она была выполнена ранее [62]. Это достигается за счёт сохранения информации о событиях исполнения: времени, номере узла и состоянии фрагментов. Такой подход устраняет необходимость динамического принятия решений системой LuNA во время исполнения и может использоваться для повышения повторяемости, воспроизводимости и производительности. В данной работе трассировка использовалась для запуска программы по заранее сформированной последовательности исполнения фрагментов. Это позволило устранить

накладные расходы, связанные с планированием исполнения. Трассы также использовались при сравнении производительности различных реализаций алгоритма.

3.5 Включение алгоритма в базу активных знаний

Разработанный фрагментированный алгоритм численного решения задачи неравновесной фильтрации реализован в виде, совместимом с архитектурой активных знаний системы LuNA. Это обеспечило возможность его включения в базу активных знаний, с дальнейшим автоматическим применением в других задачах того же класса без необходимости ручной адаптации или переписывания кода.

Согласно концепции, лежащей в основе системы LuNA, активное знание представляет собой формализованную структуру, содержащую не только код, но и семантические зависимости, описание входных и выходных данных, а также условия применения. В отличие от традиционного пассивного хранения знаний, активное знание способно интерпретироваться и использоваться системой автоматически для построения программного решения по формальному описанию задачи.

Фрагментированный алгоритм был представлен в виде совокупности вычислительных модулей (фрагментов вычислений и фрагментов данных) с явно заданными зависимостями. Такой формат соответствует требованиям, предъявляемым к элементам базы активных знаний LuNA. Каждый модуль включает информацию о типах и связях переменных, что позволяет планировщику LuNA выполнять автоматическую сборку параллельной программы на основании текущей задачи.

Благодаря включению в базу активных знаний, алгоритм может быть многократно переиспользован для решения широкого класса задач, описываемых аналогичными математическими моделями, включая задачи фильтрации, теплопереноса и многокомпонентного массообмена. Кроме того, фрагментированное представление и независимость от конкретной архитектуры исполнения обеспечивают возможность переноса решения на различные вычислительные платформы без модификации исходного алгоритма.

Таким образом, включение разработанного алгоритма в базу активных знаний LuNA повышает его воспроизводимость, масштабируемость и практическую применимость, а также способствует развитию автоматизированных средств синтеза параллельных программ для сложных научных и инженерных задач.

4 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ АЛГОРИТМОВ

Была решена одномерная задача моделирования течения многофазной жидкости (композиционная модель) с тремя фазами (нефть-газ-вода). Задача решена полным неявным методом Newton-ILU(0)-GMRES. Было произведено тестирование разработанных параллельных и фрагментированных алгоритмов. Было произведено сравнение реализованных последовательного алгоритма на языке C++, параллельного алгоритма на языке C++ с помощью стандарта MPI, параллельного фрагментированного алгоритма реализованного на языке LuNA. Характеристики матриц, размер входных данных на которых проводились тесты, показаны в таблице 1.

Таблица 1 – Размер входных данных

Номер матрицы	#1	#2	#3	#4	#5	#6	#7
Размер сетки	127	252	502	1002	2002	4002	8002
Размер матрицы	750x750	1500x1500	3000x3000	6000x6000	12000x12000	24000x24000	48000x48000

Проведены тесты последовательного алгоритма метода GMRES с предобуславливателем и без на одном узле для сравнительного анализа времени выполнения, количества итераций нужных для сходимости. Результаты данных тестов можно увидеть на рисунках 14 и 15.

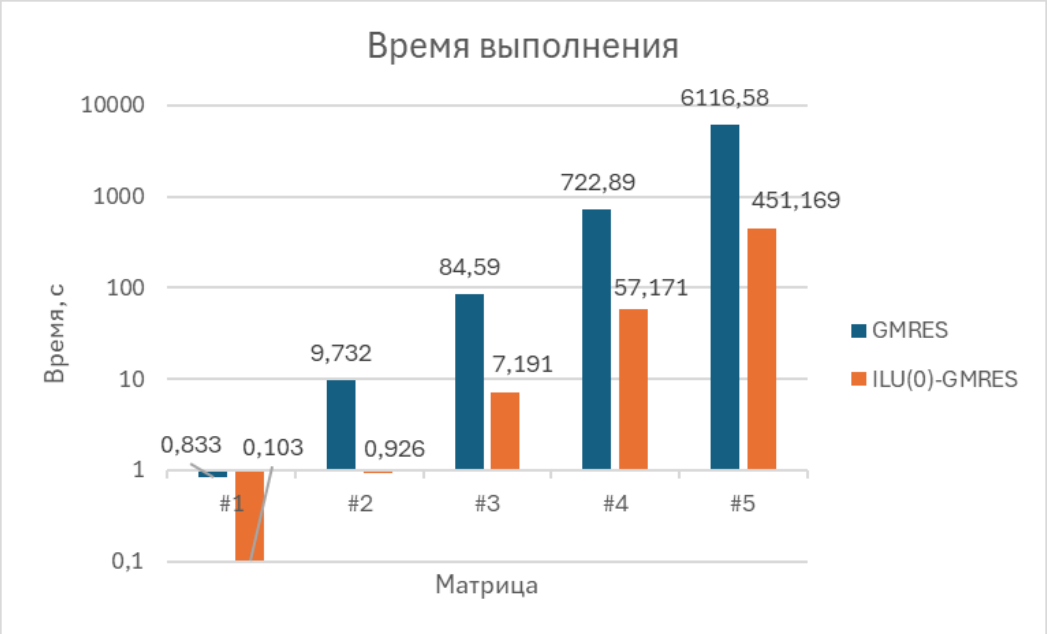


Рисунок 14 – Время выполнения последовательных алгоритмов

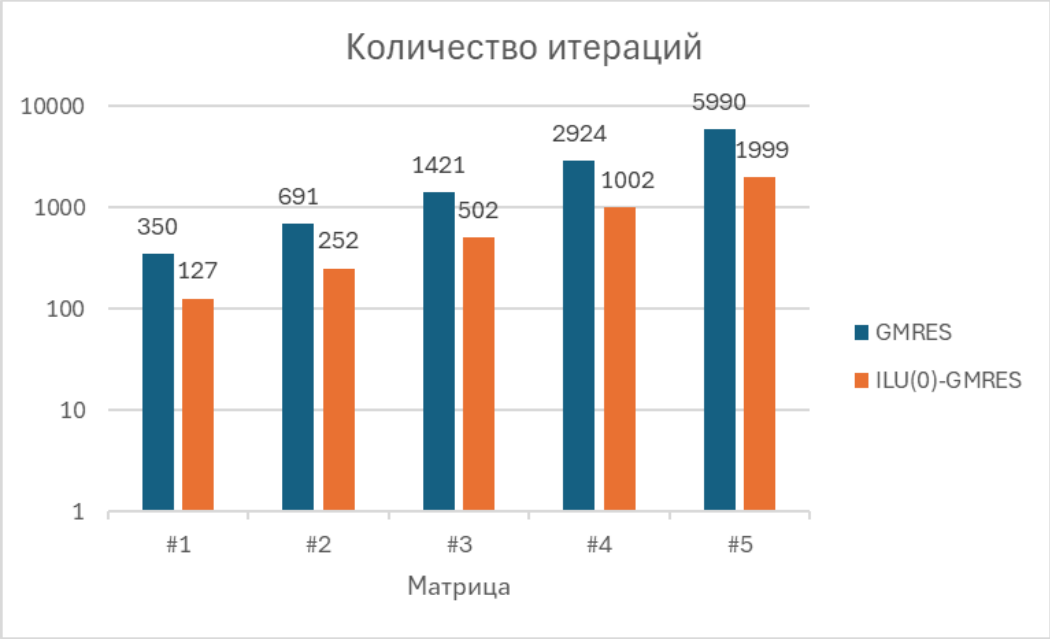


Рисунок 15 – Количество итераций необходимых для сходимости

Из результатов видно, что включение предобуславливателя уменьшает количество итераций в 3 раза, ускоряет программу в 12-13 раз на больших размерах.

Также можно увидеть невязку системы.

Таблица 2 – Невязка при решении с и без предобуславливания

	#1	**#2**	**#3**	**#4**	**#5**
GMRES	0,000946	0,000987	0,000953	0,000931	0,000987
ILU(0)-GMRES	0,000596	0,000651	0,000608	0,000492	0,000998

Данные результаты подтверждают эффективность предобуславливания при решении систем линейных уравнений методом GMRES. Существенное снижение числа итераций и времени вычислений делает данный подход предпочтительным. В связи с этим в дальнейшем при исследовании параллельной реализации будет использоваться только предобусловленный вариант алгоритма.

Далее показаны результаты параллельного алгоритма, реализованного с помощью MPI. Тесты проводились на суперкомпьютере Мвс-10п Межведомственного суперкомпьютерного центра Российской академии наук.

Таблица 3 – Время выполнения параллельного алгоритма с MPI, с

Количество процессоров	#3	#4	#5	#6	#7
1	3.74	48.38	334.24	1688.74	20219.34

Продолжение таблицы 3

Количество процессоров	#3	#4	#5	#6	#7
8	1.19	7.57	53.82	394.74	2942.02
64	2.59	12.03	64.16	389.68	2565.59
128	2.95	12.66	65.25	395.58	2569.01
256	4.39	17.29	82.75	450.28	2798.32
512	11.53	42.56	202.01	866.98	4185.07

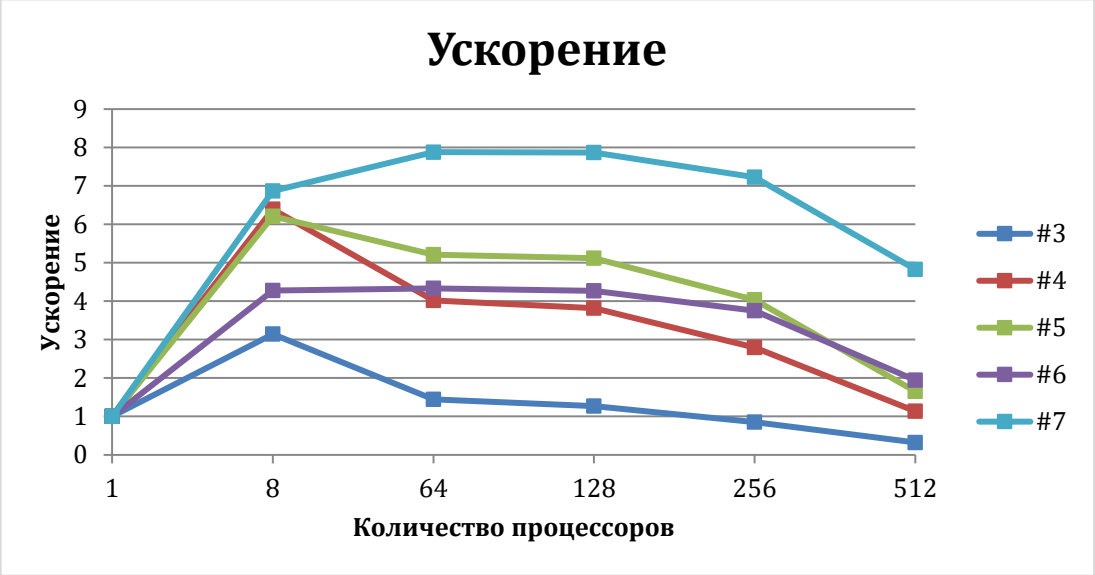


Рисунок 16 – Ускорение параллельного алгоритма с MPI

Для оценки эффективности параллельного варианта алгоритма GMRES с использованием MPI были проведены эксперименты при различном количестве процессов (от 1 до 512) и на задачах разного размера (обозначены как #3–#7). В таблице представлены времена выполнения, а на графике — соответствующее ускорение по сравнению с последовательным запуском.

Как видно из результатов, при увеличении числа процессов наблюдается значительное ускорение вплоть до определённого порога (обычно 8 или 64 процессов в зависимости от размера задачи). Однако при дальнейшем увеличении числа процессов ускорение снижается, что обусловлено возрастанием накладных расходов на межпроцессное взаимодействие (коммуникации).

Также стоит отметить, что для задач большего размера (например, #6 и #7) достигается более высокое ускорение, особенно при использовании 8 и 64 процессов. Это объясняется тем, что с ростом размера задачи вычислительная нагрузка на каждый процесс возрастает, и влияние коммуникаций становится менее существенным относительно общего времени выполнения. В результате параллельный алгоритм становится более эффективным.

Разработанный параллельный алгоритм демонстрирует хорошее масштабирование при разумном числе процессов и особенно эффективен на задачах большого размера. Ускорение ограничено увеличением коммуникационных затрат при большом числе процессов, что приводит к снижению эффективности.

Таким образом, предлагаемый алгоритм целесообразно применять на вычислительных системах с умеренным числом процессов и для решения задач большого масштаба, где достигается наибольший прирост производительности.

Далее были произведены дополнительные исследования для выявления влияния количество коммуникаций на эффективность параллельного алгоритма. В алгоритме GMRES основная нагрузка лежит на итерациях Арнольди, из-за наличия вложенных циклов, которые при параллельном выполнении требуют коммуникационных затрат на каждой итерации. При параллельном выполнении алгоритма, для вычисления вектора w, понадобится передача значения $h_{k,i}$, на каждый процессор, так как в данных вычислениях присутствует явная зависимость данных. Временная затрата усугубляется тем, что данные вычисления повторяются i раз, то есть чем больше итераций понадобятся для сходимости задачи, тем больше коммуникаций будут выполняться для каждой итераций. Таким образом, при запуске программы на большом количестве вычислительных узлов, наблюдалось замедление параллельной программы. Была протестирована параллельная программа, в котором итерации Арнольди выполняются последовательно на главном вычислительном узле. Данное решение дает ускорить работу программы при использовании библиотеки MPICH, так как количество коммуникации уменьшается. При использовании библиотеки Intel MPI, работа программы в целом замедляется, но можно наблюдать ускорение с увеличением количества вычислительных узлов.

Для определения характеристик параллельного алгоритма, в частности влияния числа коммуникаций, были разработаны и протестированы несколько версий параллельных программ, в которых различные части вычислений выполнялись последовательно на главном узле. В таблице ниже приведены сравнительные результаты выполнения программ.

Таблица 4 – Результаты тестов

	Intel			MPICH		
Количество процессов	Параллельное выполнение	Последовательный вложенный цикл	Последовательные итерации Арнольди	Параллельное выполнение	Последовательный вложенный цикл	Последовательные итерации Арнольди
8	53,82	332,13	150,79	46,16	57,49	26,01
16	51,41	246,34	152,18	34,007	40,309	26,32
32	74,92	198,37	151,39	1026,93	371,78	168,24
64	53,94	176,46	151,41	2242,83	916,83	300,26

В первом столбце для каждой библиотеки, указаны время выполнения программ, которые выполнялись полностью параллельно. В следующем столбце можно увидеть время работ программ, в которых вложенные циклы итераций Арнольди, нахождение $h_{k,i}$ и w выполнялись последовательно. В третьем столбце для каждой библиотеки, показаны время работ, когда итерации Арнольди полностью выполнялись последовательно на одном узле, а параллельно выполнялись только шаги инициазизации и нахожение приближенного решения в конце работы алгоритма.

Результаты демонстрируют, что количество коммуникаций существенно влияет на производительность параллельной программы, особенно при использовании библиотеки MPICH. Последовательное выполнение части вычислений позволяет достичь ускорения в случае MPICH, но, напротив, приводит к замедлению при использовании Intel MPI. Это связано с особенностями реализации коммуникаций: при увеличении числа процессов до 32 и 64, программа, собранная с MPICH, значительно замедляется, поскольку коммуникации начинают выполняться между узлами через TCP/IP. В то же время Intel MPI использует более быстрый InfiniBand, благодаря чему подобного замедления не наблюдается.

Далее проведены тесты фрагментированного алгоритма на языке LuNA. В таблице ниже показаны результаты тестов на матрице #1.

Таблица 5 – Результаты тестов, время в секундах. nf – количество фрагментов, np – количество процессов

nf / **np**	**2**	**4**	**8**	**16**	**32**
1	3,259	5,774	10,113	18,221	36,436
2	3,738	15,37	21,8	43,9	93,907
4	5,739	8,97	23,72	46,15	68,27
8	6,105	12,91	16,22	44,65	72,35
16	9,996	20,409	62,09	51,07	124,65
32	70,674	116,203	177,3	330,29	652,37
64	84,198	121,76	208,33	426,58	804,28
128	78,215	122,102	224,53	550,8	937,47

Одной из отличительных особенностей реализации на языке LuNA является возможность выполнения параллельной программы на разном количестве фрагментов и разном количестве процессов. Например, при реализации с MPI, данные разделяются на равное количество с количеством процессов. Большое количество фрагментов добавляет системе вариативность при построении параллельной программы, при этом это увеличивает накладные расходы, такие как где располагать данные и где проводить вычисления. В нашем случае лучшие результаты получаются при разделении всех данных на два ФД. С таблицы 4 также можно наблюдать резкое увеличение времени на 32 процессах. В этом случае мы выходим за границы одного узла, и межузловые

коммуникации происходят с посредством Ethernet, так как при сборке LuNA используется библиотека MPICH.

Далее посмотрим сравнение результатов фрагментированного алгоритма, реализованного на языке LuNA и с помощью стандарта MPI.

Таблица 6 – Результаты тестов, время в секундах

Matrix	**#1**		**#2**		**#3**		**#4**	
NP	*MPI*	*LuNA*	*MPI*	*LuNA*	*MPI*	*LuNA*	*MPI*	*LuNA*
Seq	0,103		0,926		7,191		57,171	
2	0,087	3,738	0,543	26,78	4,68	296,49	36,804	8073,31
4	0,079	5,739	0,498	39,37	4,12	366,47	27,83	7966,75
8	0,076	6,105	0,483	43,41	3,155	391,39	25,67	8751,37
16	0,199	9,996	0,735	66,06	4,57	585,02	30,77	11890,97

Как видно, реализация с системой LuNA сильно уступает по производительности. Накладные расходы при конструировании параллельных программ занимают большое время. Для оптимизации работы был использован механизм уничтожающего потребления. В фрагментированной программе ФД являются иммутабельными, то есть не изменяются после инициализации. Для изменения некой части ФД все данные копируются на новый участок памяти, что также влияет на производительность программы. Механизм уничтожающего потребления позволяет «захватить» участки памяти старого ФД без необходимости копирования. Далее показаны результаты программы с уничтожающим потреблением.

Таблица 7 – Результаты тестов с уничтожающим потреблением, время в секундах

Matrix	**#1**			**#2**			**#3**			**#4**		
NP	*MPI*	*LuNA*	*LuNA (dest. cons.)*	*MPI*	*LuNA*	*LuNA (dest. cons.)*	*MPI*	*LuNA*	*LuNA (dest. cons.)*	*MPI*	*LuNA*	*LuNA (dest. cons.)*
Seq	0,103			0,926			7,191			57,171		
2	0,087	3,74	3,49	0,543	26,78	32,98	4,68	296,49	213,03	36,804	8073,31	4401,62
4	0,079	5,74	5,39	0,498	39,37	31,64	4,12	366,47	275,55	27,83	7966,75	4828,41
8	0,076	6,11	5,87	0,483	43,41	34,03	3,16	391,39	281,19	25,67	8751,37	5317,72
16	0,199	10	11,54	0,735	66,06	59,03	4,57	585,02	429,43	30,77	11891	7855,12

Как видно из результатов, при маленьких размерах не видно сильной разницы, но при этом с увеличением размера ускорение программы ближе к двум. Это объясняется тем, что чем больше размер, тем больше времени тратится

на копирование данных, уничтожающее потребление позволяет решить как раз эту проблему.

Следующий вид оптимизации - использование механизма воспроизведения трасс. Суть данного механизма, данные о ФД и ФВ, где они выполнялись и хранились записываются в журнал событий. При повторном запуске программы эти данные воспроизводятся, и система выполнения не расходует время на такие вычисления. Ниже показаны результаты программы с воспроизведением трасс.

Таблица 8 – Результаты тестов с воспроизведением трасс, время в секундах

Matrix	**#1**			**#2**			**#3**			**#4**		
NP	*MPI*	*LuNA*	*LuNA (trace.)*	*MPI*	*LuNA*	*LuNA (trace.)*	*MPI*	*LuNA*	*LuNA (trace.)*	*MPI*	*LuNA*	*LuNA (trace.)*
Seq	0,103			0,926			7,191			57,171		
2	0,087	3,74	3,96	0,543	26,78	5,57	4,68	296,49	95,81	36,804	8073,31	1517,654
4	0,079	5,74	2,18	0,498	39,37	16,94	4,12	366,47	168,27	27,83	7966,75	3078,793
8	0,076	6,11	2,202	0,483	43,41	21,22	3,16	391,39	281,487	25,67	8751,37	5061,385
16	0,199	10	3,978	0,735	66,06	35,311	4,57	585,02	410,529	30,77	11891	7650,44

Как видно из результатов, время исполнения программы уменьшилось более чем в два раза, и данные результаты являются самыми оптимальными при реализации программы на языке LuNA.

Несмотря на то, что реализация фрагментированного алгоритма в системе LuNA в текущем виде уступает по времени исполнения реализации на MPI, она обладает существенной научной и практической ценностью. Алгоритм был разработан и реализован в фрагментированной форме, и его успешная работа в среде MPI продемонстрировала корректность, масштабируемость и преимущество по сравнению с последовательной реализацией. Реализация на LuNA подтвердила возможность автоматического исполнения фрагментированной структуры без ручного распределения вычислений. Уже в текущей версии за счёт встроенных механизмов оптимизации, таких как уничтожающее потребление и воспроизведение трасс, удалось достичь ускорения, что демонстрирует потенциал развития системы исполнения и компилятора LuNA.

Ключевая ценность предложенного решения заключается в том, что алгоритм оформлен в виде активного знания, пригодного для повторного применения. Такая форма представления обеспечивает возможность его использования не только для конкретной задачи моделирования, но и для широкого класса задач, описываемых схожими системами уравнений. Кроме того, включение алгоритма в базу активных знаний LuNA делает его

переносимым между различными вычислительными архитектурами, включая перспективные кластеры и гибридные платформы, без необходимости переписывания или адаптации исходного кода. Таким образом, даже при отставании по производительности на текущем этапе, фрагментированная реализация демонстрирует значительные перспективы в контексте автоматизации и универсализации параллельных вычислений.

ЗАКЛЮЧЕНИЕ

В работе рассмотрены современные численные методы и средства параллельной реализации, применяемые для моделирования течения многокомпонентной многофазной жидкости в пористой среде. Проведён анализ существующих подходов к численному решению системы нелинейных уравнений, возникающей при описании задачи неравновесной фильтрации, включая итерационные методы и методы предобуславливания. Показано, что наиболее эффективным в данном контексте является применение метода Ньютона для линеаризации нелинейной системы с последующим решением возникающих СЛАУ методом GMRES с предобуславливанием ILU(0).

Построен численный алгоритм, реализующий указанный подход и выполнена его параллельная реализация двумя способами: на основе технологии MPI и в виде фрагментированного алгоритма в системе фрагментированного программирования LuNA. Реализация на MPI показала преимущество разработанного алгоритма по сравнению с последовательным исполнением, а также подтвердила корректность и вычислительную эффективность выбранной численной схемы.

В целях эффективного использования памяти при работе с разреженными матрицами в алгоритме был применён компактный формат хранения CSR (Compressed Sparse Row), что позволило существенно сократить объём памяти за счёт исключения хранения нулевых элементов. Это оказалось особенно важным при решении задач крупного масштаба. Параллельная реализация позволила распределить данные между вычислительными узлами, что обеспечило возможность обработки задач, превышающих объём памяти одного узла, и продемонстрировало масштабируемость алгоритма.

Фрагментированная реализация алгоритма обеспечила возможность использования системных средств автоматического управления ресурсами и выполнением в среде LuNA. В частности, в программную реализацию были встроены механизмы уничтожающего потребления и воспроизведения трасс, что позволило снизить накладные расходы и повысить повторяемость выполнения. Разработанный фрагментированный алгоритм добавлен в базу активных знаний системы LuNA, что обеспечивает его повторное применение для решения других задач аналогичной структуры без необходимости ручной модификации кода.

Сравнительный анализ реализации на MPI и LuNA показал, что несмотря на некоторое отставание по времени исполнения, фрагментированная реализация обладает существенными преимуществами: переносимость, автоматическое управление исполнением, возможность повторного использования и масштабируемость. Эти свойства делают её перспективной платформой для дальнейших исследований и применения в задачах моделирования сложных физических процессов.

Таким образом, в диссертационной работе разработан и реализован эффективный фрагментированный алгоритм численного решения задачи неравновесной фильтрации, пригодный для включения в инфраструктуру

активных знаний и автоматизированного исполнения в среде LuNA. Полученные результаты подтверждают применимость и эффективность предложенного подхода в задачах научного моделирования с использованием современных параллельных вычислений.

Диссертационная работа выполнялась в рамках научно-исследовательской работы проектов грантового финансирования «Разработка системы управления активными знаниями для автоматизации конструирования высокопроизводительных параллельных программ обработки неструктурированных данных и численного моделирования в задачах фильтрации» в ДГП НИИ математики и механики и «Высокопроизводительное компьютерное моделирование движения многофазной жидкости в пористой среде в условиях неопределенности» в РОО «НИА РК». Результаты исследований по данной диссертации включены в отчеты этих проектов.

Также основные результаты этой диссертационной работы отражены в следующих публикациях:

- Mukhambetzhanov, S. T., Lebedev D.V., Kassymbek N.M., Imankulov T.S., Matkerim B., Akhmed-Zaki D.Zh. (2020). GMRES based numerical simulation and parallel implementation of multicomponent multiphase flow in porous media. Cogent Engineering, 7(1), 1785189. https://doi.org/10.1080/23311916.2020.1785189
- Imankulov T.S., Kassymbek N.M., Lebedev D.V., Matkerim B., Daribayev B.S. Numerical Simulation of Multiphase Multicomponent Flow in Porous Media: Efficiency Analysis of Newton-Based Method. Fluids 2021, 6, 355. https://doi.org/10.3390/fluids6100355
- Қасымбек Н.М., Ахмед-Заки Д.Ж., Иманкулов Т.С., Лебедев Д.В., Кенжебек Е.Ғ. Highperformance and Intelligent Computational Models for Oil Production Problems // Вестник НИА РК 2020 - г. 8 - стр.
- Қасымбек Н.М., Лебедев Д.В. МОДЕЛИРОВАНИЕ ДВИЖЕНИЯ ДВУХФАЗНОЙ ЖИДКОСТИ С ПОМОЩЬЮ МЕТОДА НЬЮТОНА // ВЕСТНИК КазНПУ им. Абая, серия «Физико-математические науки», No4(76), 2021 г. с. 38-44
- Касымбек Н.М., МаткеримБ., Лебедев Д.В., Иманкулов Т.С., Ахмед-Заки Д.Ж., Численное моделирование и параллельная реализация многокомпонентного многофазного течения в пористых средах на основе метода GMRES. МАТЕРИАЛЫ IV международной научно-практической конференции "Информатика и прикладная математика", 25-29 сентябрь 2019, Алматы, Казахстан. - С. 253 -260.
- N. Kassymbek, B. Matkerim, D. Lebedev, T. Imankulov and D. Akhmed-Zaki, GMRES Based Numerical Simulation of Multicomponent Multiphase Flow in Porous Media on LuNA Fragmented Programming System, Conference Proceedings, ECMOR XVII, Sep 2020, Volume 2020, p. 1-10.
- Ахмед-Заки Д.Ж., Лебедев Д.В., Қасымбек Н.М. Высокопроизводительное моделирование движения многокомпонентной многофазной жидкости в пористой среде / Традиционная международная апрельская математическая конференция в честь Дня работников науки

Республики Казахстан, посвященная 75-летию академика НАН РК Тынысбека Шариповича Кальменова", Международный, КАЗАХСТАН, 05.04-08.04.2021

- Kassymbek, N., Lebedev, D., & Akhmed-Zaki, D. (2022). Optimization of a GMRES-Based Parallel Algorithm for Numerical Simulation of Multicomponent Multiphase Flow in Porous Media. In ECMOR 2022. ECMOR 2022. European Association of Geoscientists & Engineers. https://doi.org/10.3997/2214-4609.202244055

Список литературы

1 Pedersen K. S., Christensen P. L., Shaikh J. A. Phase Behavior of Petroleum Reservoir Fluids: 2-е изд. - Boca Raton: CRC Press, 2014. - 465 с. - ISBN 978-1439852279. - DOI 10.1201/b17887.

2 Ahmed T. Reservoir Engineering Handbook: 3-е изд. - Burlington (MA); Oxford: Gulf Professional Publishing (Elsevier), 2006. - 1376 с. - ISBN 0-7506-7972-7. - URL: https://shop.elsevier.com/books/reservoir-engineering-handbook/ahmed/978-0-7506-7972-5 (дата обращения: 28.05.2025).

3 Borisov V. E., Kritskiy B. V., Marchenko N. A., Mitrushkin D. A., Savenkov E. B. Non-isothermal compositional flow model with chemical reactions and active solid phase for reservoir simulation: препринт № 91. - Moscow: Keldysh Institute of Applied Mathematics, 2013. - URL: https://library.keldysh.ru/preprint.asp?id=2013-91 (дата обращения: 08.08.2025).

4 Delshad M., Thomas S. G., Wheeler M. F. Parallel numerical reservoir simulations of nonisothermal compositional flow and chemistry // SPE Journal. - 2011. - Т. 16, № 2. - С. 239-248. - DOI 10.2118/118847-PA.

5 Chemical Reaction Modeling in a Compositional Reservoir-Simulation Framework // SUETRI-B Reservoir Simulation (Stanford University). - 2013. - URL: https://suetri-b.stanford.edu/publications/conference-papers/chemical-reaction-modeling-compositional-reservoir-simulation (дата обращения: 28.05.2025).

6 Maurand N., Mayoura S., Parra T., Araujo C. H. V., Furtado C. J. A. Coupling compositional flow, thermal effects and geochemistry reactions when injecting CO2 in a carbonated oil field // Energy Procedia. - 2014. - Т. 51. - С. 316-325. - DOI 10.1016/j.egypro.2014.07.038.

7 Chen Z. Reservoir Simulation: Mathematical Techniques in Oil Recovery. - Philadelphia (PA): SIAM, 2007. - 248 с. - (CBMS-NSF Regional Conference Series in Applied Mathematics; 77). - ISBN 978-0-89871-640-5.

8 Chen Z., Huan G., Wang H. Computer simulation of compositional flow using unstructured control volume finite element methods // Computing. - 2006. - Т. 78, № 1. - С. 31-53. - DOI 10.1007/s00607-006-0171-5.

9 Aceto L., Pandolfi R., Trigiante D. One parameter family of linear difference equations and the stability problem for the numerical solution of ODEs // Advances in Difference Equations. - 2006. - Article ID 19276. - 14 с. - DOI 10.1155/ADE/2006/19276.

10 Edwards L., Dhanpat D., Chakrabarti D. P. Hydrodynamics of three phase flow in upstream pipes // Cogent Engineering. - 2018. - Т. 5, № 1. - Ст. 1433983. - DOI 10.1080/23311916.2018.1433983.

11 Khramchenkov E., Khramchenkov M. Numerical model of two-phase flow in dissolvable porous media and simulation of reservoir acidizing // Natural Resources Research. - 2018. - Т. 27, № 4. - С. 531-537. - DOI 10.1007/s11053-018-9371-x.

12 Вержбицкий В. М. Численные методы. Математический анализ и обыкновенные дифференциальные уравнения: учеб. пособие. - Санкт-Петербург:

СПбПУ, 2021. - 288 с. - URL: https://elib.spbstu.ru/dl/2/ek22-7.pdf/en/info (дата обращения: 28.05.2025).

13 Higham N. J. Gaussian elimination // WIREs Computational Statistics. - 2011. - Т. 3, № 3. - С. 230-238. - DOI 10.1002/wics.164.

14 Saad Y. Iterative Methods for Sparse Linear Systems. - 2nd ed. - Philadelphia (PA): SIAM, 2003. - 528 с. - (Other Titles in Applied Mathematics; 82). - ISBN 978-0-89871-534-7. - DOI 10.1137/1.9780898718003.

15 Van der Vorst H. A. Iterative Krylov Methods for Large Linear Systems. - Cambridge: Cambridge University Press, 2003. - 228 с. - (Cambridge Monographs on Applied and Computational Mathematics; 13). - ISBN 978-0-521-55267-5. - DOI 10.1017/CBO9780511615115.

16 Vinsome P. K. W. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations // SPE Symposium on Numerical Simulation of Reservoir Performance. - 1975. - DOI 10.2118/5729-MS. - URL: https://dx.doi.org/10.2118/5729-MS (дата обращения: 28.05.2025).

17 Saad Y., Schultz M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems // SIAM Journal on Scientific and Statistical Computing. - 1986. - Т. 7, № 3. - С. 856-869. - DOI 10.1137/0907058.

18 Lacroix S., Vassilevski Yu., Wheeler J., Wheeler M. Iterative solution methods for modeling multiphase flow in porous media fully implicitly // SIAM Journal on Scientific Computing. - 2003. - Т. 25, № 3. - С. 905-926. - DOI 10.1137/S106482750240443X.

19 Mittal R. C., Al-Kurdi A. LU-decomposition and numerical structure for solving large sparse nonsymmetric linear systems // Computers & Mathematics with Applications. - 2002. - Т. 43, № 1. - С. 131-155. - DOI 10.1016/S0898-1221(01)00279-6.

20 Al-Kurdi A., Kincaid D. R. LU-decomposition with iterative refinement for solving sparse linear systems // Journal of Computational and Applied Mathematics. - 2006. - Т. 185, № 2. - С. 391-403. - DOI 10.1016/j.cam.2005.03.018.

21 Chow E., Saad Y. ILUS: An incomplete LU preconditioner in sparse skyline format // International Journal for Numerical Methods in Fluids. - 1997. - Т. 25, № 7. - С. 739-748. - DOI 10.1002/(SICI)1097-0363(19971015)25:7<739::AID-FLD581>3.0.CO;2-Y.

22 Chow E., Saad Y. Experimental study of ILU preconditioners for indefinite matrices // Journal of Computational and Applied Mathematics. - 1997. - Т. 86, № 2. - С. 387-414. - DOI 10.1016/S0377-0427(97)00171-4.

23 Mittal R. C., Al-Kurdi A. H. An efficient method for constructing an ILU preconditioner for solving large sparse nonsymmetric linear systems by the GMRES method // Computers & Mathematics with Applications. - 2003. - Т. 45, № 10. - С. 1757-1772. - DOI 10.1016/S0898-1221(03)00154-8.

24 Gropp W., Lusk E., Doss N., Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard // Parallel Computing. - 1996. - Т. 22, № 6. - С. 789-828. - DOI 10.1016/0167-8191(96)00024-5.

25 Андонов А. С. Параллельное программирование с использованием технологии OpenMP // PARALLEL.RU - Информационно-аналитический центр по параллельным вычислениям. - URL: https://parallel.ru/info/parallel/openmp (дата обращения: 28.05.2025).

26 Danaev N., Akhmed-Zaki D., Mukhambetzhanov S., Imankulov T. Mathematical modelling of oil recovery by polymer/surfactant flooding // In: Mathematical Modeling of Technological Processes / Danaev N., Shokin Y., Darkhan A.-Z. (eds.). - Cham: Springer, 2015. - С. 1-12. - DOI 10.1007/978-3-319-25058-8_1.

27 Iryanto, Gunawan P. H. An OpenMP parallel Godunov scheme for 1D two-phase oil displacement problem // Proc. 5th International Conference on Information and Communication Technology (ICoICT). - 2017. - С. 1-5. - DOI 10.1109/ICoICT.2017.8074664.

28 Werneck L., Freitas M., Júnior H., de Souza G., Souto H. An OpenMP parallel implementation for numerical simulation of gas reservoirs using Intel Xeon Phi coprocessor: препринт. - 2016.

29 Pannala S., D'Azevedo E., Syamlal M. Hybrid (OpenMP and MPI) parallelization of MFIX: a multiphase CFD code for modeling fluidized beds // In: Proceedings of the 2003 ACM Symposium on Applied Computing (SAC'03). - New York: ACM, 2003. - С. 199-206. - DOI 10.1145/952532.952574.

30 Akhmed-Zaki D., Daribayev B., Imankulov T., Turar O. High-performance computing of oil recovery problem on a mobile platform using CUDA technology // Eurasian Journal of Mathematical and Computer Applications. - 2017. - Т. 5, № 2. - С. 4-13. - DOI 10.32523/2306-6172-2017-5-2-4-13.

31 Zaza A., Awotunde A. A., Fairag F. A., Al-Mouhamed M. A. A CUDA-based parallel multi-phase oil reservoir simulator // Computer Physics Communications. - 2016. - Т. 206. - С. 2-16. - DOI 10.1016/j.cpc.2016.04.010.

32 McClure J. E., Prins J. F., Miller C. T. A novel heterogeneous algorithm to simulate multiphase flow in porous media on multicore CPU-GPU systems // Computer Physics Communications. - 2014. - Т. 185, № 7. - С. 1865-1874. - DOI 10.1016/j.cpc.2014.03.012.

33 Foster I., Kesselman C., Tuecke S. The anatomy of the Grid: enabling scalable virtual organizations // The International Journal of High Performance Computing Applications. - 2001. - Т. 15, № 3. - С. 200-222. - DOI 10.1177/109434200101500302.

34 Allcock B., et al. Data management and transfer in high-performance computational grid environments // Parallel Computing. - 2002. - Т. 28, № 5. - С. 749-771. - DOI 10.1016/S0167-8191(02)00094-7.

35 BLAS (Basic Linear Algebra Subprograms). - URL: https://www.netlib.org/blas/ (дата обращения: 28.05.2025).

36 LAPACK - Linear Algebra PACKage. - URL: https://www.netlib.org/lapack/ (дата обращения: 28.05.2025).

37 Balay S., et al. PETSc/TAO Users Manual. Revision 3.22. - Argonne National Laboratory, 2024. - (ANL--21/39-Rev-3.22). - DOI 10.2172/2476320.

38 Charm++: Parallel Programming Framework. - URL: https://charmplusplus.org/ (дата обращения: 28.05.2025).

39 Li S., Zhang C.-S. OpenCAEPoro: A parallel simulation framework for multiphase and multicomponent porous media flows: preprint // arXiv. - 2024-06-16. - arXiv:2406.10862. - DOI 10.48550/arXiv.2406.10862.

40 Malyshkin V. E., Perepelkin V. A. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem // In: Parallel Computing Technologies. - Berlin, Heidelberg: Springer, 2011. - С. 53-61. - DOI 10.1007/978-3-642-23178-0_5.

41 Akhmed-Zaki D., Lebedev D., Malyshkin V., Perepelkin V. Automated construction of high performance distributed programs in LuNA system // In: Parallel Computing Technologies. - Cham: Springer, 2019. - С. 3-9. - DOI 10.1007/978-3-030-25636-4_1.

42 Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a three-dimensional three-phase fluid flow (“oil-water-gas”) numerical model in LuNA fragmented programming system // The Journal of Supercomputing. - 2017. - Т. 73, № 2. - С. 624-630. - DOI 10.1007/s11227-016-1780-1.

43 Belyaev N., Perepelkin V. High-efficiency specialized support for dense linear algebra arithmetic in LuNA system // In: Parallel Computing Technologies. - Cham: Springer, 2021. - С. 143-150. - DOI 10.1007/978-3-030-86359-3_11.

44 Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed algorithm of data allocation in the fragmented programming system LuNA // In: Parallel Computing Technologies. - Cham: Springer, 2015. - С. 80-85. - DOI 10.1007/978-3-319-21909-7_8.

45 Александрович К. А., Малышкин В. Э., Юрьевич Н. Ю., Александрович П. В., Андреевич С. В. Эффективная фрагментированная реализация краевой задачи фильтрации двухфазной жидкости // Проблемы информатики. - 2023. - Вып. 2 (59).

46 Dogru A. H., Sunaidi H. A., Fung L. S., Habiballah W. A., Al-Zamel N., Li K. G. A parallel reservoir simulator for large-scale reservoir simulation // SPE Reservoir Evaluation & Engineering. - 2002. - Т. 5, № 1. - С. 11-23. - DOI 10.2118/75805-PA.

47 Бугров А. Н. Об организации параллельных вычислений при решении разностных уравнений // Системный анализ в науке и образовании. - 2016. - Вып. 1.

48 Tian M., Liu Q., Pan J., Gou Y., Zhang Z. swPTS: an efficient parallel Thomas split algorithm for tridiagonal systems on Sunway manycore processors // The Journal of Supercomputing. - 2024. - Т. 80, № 4. - С. 4682-4706. - DOI 10.1007/s11227-023-05641-1.

49 Bai Z., Hu D., Reichel L. A Newton basis GMRES implementation // IMA Journal of Numerical Analysis. - 1994. - Т. 14, № 4. - С. 563-581. - DOI 10.1093/imanum/14.4.563.

50 Erhel J. A parallel GMRES version for general sparse matrices // Electronic Transactions on Numerical Analysis. - 1995. - Т. 3. - С. 160-176.

51 Wakam D., Guy Antoine A. K. Parallel GMRES with a multiplicative Schwarz preconditioner // ARIMA - Revue Africaine de Recherche en Informatique et Mathématiques Appliquées. - 2010. - Спец. вып. Volume 14-2011-Special. - DOI 10.46298/arima.1945.

52 Ghysels P., Ashby T. J., Meerbergen K., Vanroose W. Hiding global communication latency in the GMRES algorithm on massively parallel machines // SIAM Journal on Scientific Computing. - 2013. - Т. 35, № 1. - С. C48-C71. - DOI 10.1137/12086563X.

53 Calandra H., Gratton S., Lago R., Vasseur X., Carvalho L. M. A modified block flexible GMRES method with deflation at each iteration for the solution of non-Hermitian linear systems with multiple right-hand sides // SIAM Journal on Scientific Computing. - 2013. - Т. 35. - С. S345-S367. - DOI 10.1137/120883037.

54 Bahi J. M., Couturier R., Khodja L. Z. Parallel sparse linear solver GMRES for GPU clusters with compression of exchanged data // In: Euro-Par 2011: Parallel Processing Workshops. - Berlin, Heidelberg: Springer, 2012. - С. 471-480. - DOI 10.1007/978-3-642-29737-3_52.

55 Accelerating linear solvers for reservoir simulation on GPU workstations // Proc. 24th High Performance Computing Symposium. - San Diego: Society for Modeling and Simulation International (SCS), 2016. - DOI 10.22360/SpringSim.2016.HPC.007.

56 Chen Z., Huan G., Ma Y. Computational Methods for Multiphase Flows in Porous Media. - Philadelphia (PA): SIAM, 2006. - (Computational Science & Engineering). - DOI 10.1137/1.9780898718942. - URL: https://epubs.siam.org/doi/book/10.1137/1.9780898718942 (дата обращения: 28.05.2025).

57 Imankulov T., Lebedev D., Matkerim B., Daribayev B., Kassymbek N. Numerical simulation of multiphase multicomponent flow in porous media: efficiency analysis of Newton-based method // Fluids. - 2021. - Т. 6, № 10. - Ст. 355. - DOI 10.3390/fluids6100355.

58 Pissanetzky S. Sparse Matrix Technology. - London: Academic Press, 1984.

59 Mukhambetzhanov S. T., Lebedev D. V., Kassymbek N. M., Imankulov T. S., Matkerim B., Akhmed-Zaki D. Zh. GMRES-based numerical simulation and parallel implementation of multicomponent multiphase flow in porous media // Cogent Engineering. - 2020. - Т. 7, № 1. - Ст. 1785189. - DOI 10.1080/23311916.2020.1785189.

60 Kassymbek N., Matkerim B., Lebedev D., Imankulov T., Akhmed-Zaki D. GMRES-based numerical simulation of multicomponent multiphase flow in porous media on LuNA fragmented programming system // In: ECMOR XVII - 17th European Conference on the Mathematics of Oil Recovery. - 2020. - С. 1-10. - DOI 10.3997/2214-4609.202035201.

61 Kassymbek N., Lebedev D., Akhmed-Zaki D. Optimization of a GMRES-based parallel algorithm for numerical simulation of multicomponent multiphase flow in porous media // In: ECMOR 2022 - 18th European Conference on

the Mathematics of Oil Recovery. - 2022. - C. 1-10. - DOI 10.3997/2214-4609.202244055.

62 Malyshkin V., Perepelkin V., Lyamin A. Trace balancing technique for trace playback in LuNA system // In: Parallel Computing Technologies. - Cham: Springer, 2023. - C. 42-50. - DOI 10.1007/978-3-031-41673-6_4.

ПРИЛОЖЕНИЕ А

Программный код реализации фрагментированного алгоритма для решения задачи неравновесной фильтрации.

```
C++ head ${<C++ end}
#include <cstdio>
#include <mpi.h>
C++ end
C++ sub mpi_comm_size(name size) ${{
  int sz;
  MPI_Comm_size(MPI_COMM_WORLD,&sz);
  size.setValue(sz);
$}}
import __set_int(name, int) as _set_int;
import c_iprint(int) as iprint;
import c_print(value) as print;
import c_printime(value, value) as printime;
import c_intprint(value) as intprint;
import sum(int, int, name) as sum;
import c_gather_arr(int, value, value, name) as gather_arr;
import copy_vector(value, name) as copy_v;
import c_setn(int, name) as setN;
import c_setnz(int, name) as setNZ;
import c_setnx(value, int, name) as setNX;
import c_setrows(value, int, value, name) as setROWS;
import c_initP(int, name) as initP;
import c_initA(int, value, value, name, name, name) as initA;
import c_initb(name, value, value, value, value, value, value, value, value, value,
value, value, value, value, value, value, value, value, value, value, value, value,
value, value, value, value, value, value, value, value, value, value, value, value,
int) as initb;
import c_scatterA(int,int,value,value,value,value,value,name,name,name) as
scatterA;
import c_scatterVector(int, int, value, value, value, name) as scatterVector;
import c_setzeroH(name, value) as setzeroH;
import c_setzerotoarray(name, value) as setzerotoarray;
import c_setEarray(name, value) as setEarray;
import c_setX0array(name, value) as setX0array;
import c_normpart(int, value, value, name) as normpart;
import c_normsum(value,value,name) as normsum;
import c_setnorm(value,name) as setnorm;
import c_setdftodf(value,name) as setdftodf;
```

```
import c_checkbnrm(value,name) as checkbnrm;
import c_r0partexec(int, value, value, value, value, value, value, value, name) as
r0partexec;
import c_initVpart1(int,value, value, value, value, name) as initVpart1;
import c_qexecute(value,name,value,value) as qexecute;
import c_setV(value,name,value) as setV;
import c_setV1(value,name,value) as setV1;
import c_execw0(int,value,value,value,value,value,name) as execw0;
import c_multipleVtow(int,value,value,value,name) as multipleVtow;
import c_execw(int,int,int,value,value,value,value,value,name) as execw;
import c_executeHjplus(value,int,value,value,name) as executeHjplus;
import c_initVpartj(int,int,value,value,value,value,name) as initVpartj;
import c_executeHi(int,int,value,value,value,name) as executeHi;
import
c_givensRotation(int,value,value,value,value,value,value,name,name,name,name,na
me) as givensRotation;
import c_calcR(int,value,value,name) as calcR;
import c_scatterMatrix(int, int, value, value, value, name) as scatterMatrix;
import c_calcypart(int,value,value,value,value,name) as calcypart;
import c_reduceV(int,int,value,value,value,name) as reduceV;
import c_calcx0(int,value,value,value,value,value,name) as calcx0;
import c_printarray(value,int) as printarray;
import c_print_array(int, value) as print_array;
import c_beginvalues(int, name, name, name, name, name, name, name, name,
name, name, name, name, name, name, name, name, name, name, name, name, name, name,
name, name, name, name, name, name, name, name, name, name, name) as
beginvalues;
import c_calcnewz2(value,int,name) as calcnewz2;
import c_valuecopy(value, name, int) as valuecopy;
import c_calcP(value,name,name,name,int) as calcP;
import c_calcavg(value,int,name) as calcavg;
import c_ilu0(value, value, value, value, value, name, name, name, name) as
ilu0;
import c_LUmatrixSeparation(value, value, value, value, value, value, name,
name, name, name, name, name) as LUmatrixSeparation;
import c_GaussSolve(value,value,value,value,int,value,name) as GaussSolve;
import c_init(int, name) as init;
import c_initime(int, name, name) as initime;
import c_finddelta(value, name, name, name, name, name, name, int) as
finddelta;
import c_findmax(value, value, value, value, value, value, int, name) as findmax;
import c_newvalue(name, value, value, int) as newvalue;
import c_newvalueS(name, value, value, int) as newvalueS;
import c_newvalueP(name, value, value, int) as newvalueP;
```

```
import c_findpair(name, value, int) as findpair;
import c_printerror(int, value) as printerror;
#define FULLN 127
#define RESTR 129
#define FG_COUNT 4
#define ITMAX 1
#define EPS 0.001
#define LOC1(i,nf,np) ($i*$np/$nf)
#define RESTR1 1
sub main()
{
    df N, NX, ROWS, NZ, A, Apart, b, bpart, oldSo, newSo, oldSg, newSg, oldSw, newSw, oldPo, newPo, oldPg, newPg, oldPw, newPw, oldP, newP, oldS, newS, oldKso, newKso, oldKsg, newKsg, oldKsw, newKsw, newz1, newz2, newz, oldz1, oldz2, oldz, oldx1o, oldx2o, newx1o, newx2o, oldx1g, oldx2g, newx1g, newx2g, oldV, oldL, newV, newL, fi1o, fi2o, fi1g, fi2g, oldF, newF, dL, dV, dF, dS, dX1o, dX1g, dP, dz1, delta, m, h, mo, mg, mw, tau, k, avg, maxdelta,L,U,ILU,uptr,np,nf; /*NX - number of lines in each data fragments
                ROWS - begin of lines
                N - matrix size
                NZ - non-zero elements
                A - matrix
                b - vector
                */
    setN($FULLN, N);
    init($FG_COUNT, nf);
    setNZ($FULLN, NZ);
    setNX(N,$FG_COUNT,NX);
    setROWS(N,$FG_COUNT,NX,ROWS);
    mpi_comm_size(np);
    beginvalues($FULLN,newx1o, newx2o, newx1g, newx2g, newL, newV, newS, newP, newz1, newz2, newF, oldKso, newKso, oldKsg, newKsg, oldKsw, newKsw, fi1o, fi2o, fi1g, fi2g, m, h, mo, mg, mw, tau, k, newSo, newSg, oldSo, oldSg);
    valuecopy(newx1o, oldx1o, $FULLN) @ {request newx1o;};
    valuecopy(newx2o, oldx2o, $FULLN) @ {request newx2o;};
    valuecopy(newL, oldL, $FULLN) @ {request newL;};
    valuecopy(newx1g, oldx1g, $FULLN) @ {request newx1g;};
    valuecopy(newx2g, oldx2g, $FULLN) @ {request newx2g;};
    valuecopy(newV, oldV, $FULLN) @ {request newV;};
    valuecopy(newS, oldS, $FULLN) @ {request newS;};
    valuecopy(newP, oldP, $FULLN) @ {request newP;};
    valuecopy(newF, oldF, $FULLN) @ {request newF;};
    calcP(newP,newPo,newPw,newPg,$FULLN) @ {request newP;};
    calcavg(newL,$FULLN-1,avg) @ {request newL;};
```

```
initA($FULLN,N,NZ,A[0],A[1],A[2]);
initb(b, newPo, newx1o, fi1o, newPg, newx1g, fi1g, fi2g, newx2o, fi2o, newx2g, newKso, newSo, newKsg, newSg, oldx1o, oldKso, oldSo, oldx1g, oldKsg, oldSg, oldS, newF, oldF, newKsw, newS, oldKsw, newPw, newL, h, m, tau, k, mo, mg, mw, $FULLN) @ {request newPo; request newx1o; request fi1o; request newPg; request newx1g; request fi1g; request fi2g; request newx2o; request fi2o; request newx2g; request newKso; request newSo; request newKsg; request newSg; request oldx1o; request oldKso; request oldSo; request oldx1g; request oldKsg; request oldSg; request oldS; request newF; request oldF; request newKsw; request newS; request oldKsw; request newPw; request newL; request h; request m; request tau; request k; request mo; request mg; request mw;};
scatter_A_matrix(N,$FG_COUNT,ROWS,A,Apart,np);
for i=0..$FG_COUNT-1
    scatterVector(i,$FG_COUNT,ROWS,N,b,bpart[i]) @ {locator_cyclic: $LOC1(i,$FG_COUNT,np); request b;};
ilu0(N, NZ, A[0], A[1], A[2], ILU[0], ILU[1], ILU[2], uptr) @ { request A[0]; request A[1]; request A[2];};
LUmatrixSeparation(N, NZ, ILU[0], ILU[1], ILU[2], uptr, L[0], L[1], L[2], U[0], U[1], U[2]) @ {request ILU[0]; request ILU[1]; request ILU[2]; request uptr;};
Gmresilu(N,NZ,NX,ROWS,Apart,bpart,L,U,delta,np);
finddelta(delta, dX1o, dL, dS, dP, dz1, dF, $FULLN) @ {delete A[0]; delete A[1]; delete A[2]; delete ILU[0]; delete ILU[1]; delete ILU[2]; delete L[0]; delete L[1]; delete L[2]; delete U[0]; delete U[1]; delete U[2]; delete b;  delete oldSo; delete newSo; delete oldSg; delete newSg; delete oldSw; delete newSw; delete oldPo; delete newPo; delete oldPg; delete newPg; delete oldPw; delete newPw; delete oldP; delete newP; delete oldS; delete newS; delete oldKso; delete newKso; delete oldKsg; delete newKsg; delete oldKsw; delete newKsw; delete newz1; delete newz2; delete newz; delete oldz1; delete oldz2; delete oldz; delete oldx1o; delete oldx2o; delete newx1o; delete newx2o; delete oldx1g; delete oldx2g; delete newx1g; delete newx2g; delete oldV; delete oldL; delete newV; delete newL; delete fi1o; delete fi2o; delete fi1g; delete fi2g; delete oldF; delete newF; delete dL; delete dV; delete dF; delete dS; delete dX1o; delete dX1g; delete dP; delete dz1; delete delta; delete m; delete h; delete mo; delete mg; delete mw; delete tau; delete k; delete avg; delete maxdelta; delete uptr;};
findmax(dX1o, dL, dS, dP, dz1, dF, $FULLN, maxdelta);
newvalue(newx1o[1], newx1o, dX1o, $FULLN);
newvalue(newL[1], newL, dL, $FULLN);
newvalueS(newS[1], newS, dS, $FULLN);
newvalueP(newP[1], newP, dP, $FULLN);
newvalue(newz[1], newz, dz1, $FULLN);
newvalue(newF[1], newF, dF, $FULLN);
findpair(newx1g[1], newx1o[1], $FULLN);
findpair(newV[1], newL[1], $FULLN);
```

```
		printarray(newP[1],$FULLN);*/
		for i=0..$FG_COUNT-1
		{
			intprint(N) @ { locator_cyclic: $LOC1(i,$FG_COUNT,np);};
		}
	} @ {
		locator_replicating N => all;
		locator_replicating NX => all;
		locator_replicating ROWS => all;
		locator_cyclic bpart[i]=>i;
		locator_cyclic Apart[j][i]=>i;
	}

	sub scatter_A_matrix(name n, int size, name rows, name A, name Apart, name
np){
		for i=0..size-1
		{

	scatterA(i,size,n,rows,A[0],A[1],A[2],Apart[0][i],Apart[1][i],Apart[2][i])  @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); request A[0]; request A[1]; request
A[2];};//A[0] - value, A[1] - col, A[2] - index
		}
	} @ {
			locator_cyclic Apart[j][i]=>i;
			locator_replicating n => all;
			locator_replicating rows => all;
	}

	sub Gmresilu(name N, name NZ, name NX, name ROWS, name Apart, name
bpart, name L, name U, name result, name np){
	//sub Gmresilu(name ROWS){
		df
V,Vpart,Vtemp,Vp,H,Htemp,spart,R,Rpart,c,s,x0,x0part,xtemp,e,q,wpart,w,wtemp,y,
ypart,ytemp,m,x,M,z,K,bnrm,betta,error,temp,r0,r0part,rtemp,norma,MX,MROWS,
time1;
		initime($RESTR, m, time1);
		setEarray(e, m);
		setX0array(x0[0],N);
		norm_execute(bpart,NX,bnrm,np);
		init(0, x[0]);
		while x[itr]<$ITMAX, itr = 0..out M{
			for i=0..$FG_COUNT-1
			{
```

```
        scatterVector(i,$FG_COUNT,ROWS,N,x0[itr],x0part[itr][i])                @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); request x0[itr];};
                        }
                        setzeroH(H[itr][0][0],m) @ { request m;};
                        setzerotoarray(c[itr][0],m) @ { request m;};
                        setzerotoarray(s[itr][0],m) @ { request m;};
                        findr0(Apart,bpart,x0[itr],r0part[itr][0],NX,ROWS,np);
                        gather_arrayDel(r0part[itr][0], $FG_COUNT, r0[itr][0],np);
                        GaussSolve(L[0],L[1],L[2],N,1,r0[itr][0],r0[itr][1])   @   {request
L[0],L[1],L[2],r0[itr][0];};
                        GaussSolve(U[0],U[1],U[2],N,0,r0[itr][1],r0[itr][2])   @   {request
U[0],U[1],U[2],r0[itr][1];};
                        for i=0..$FG_COUNT-1

        scatterVector(i,$FG_COUNT,ROWS,N,r0[itr][2],r0part[itr][1][i]);
                        norm_execute(r0part[itr][1],NX,betta[itr],np);
                        for i=0..$FG_COUNT-1
                        {
                initVpart1(i,NX,ROWS,r0part[itr][1][i],betta[itr][i],Vpart[itr][0][i])     @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                initVpart1(i,NX,ROWS,r0part[itr][1][i],betta[itr][i],Vp[itr][0][i])        @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                        }
                        qexecute(e,q[itr][0],betta[itr][0],m) @ {request e,betta[itr][0],m;};
                        init(0, z[0]);
                        while z[j]<$RESTR, j = 0..out K{ //$RESTR1
                                //intprint(z[j]);
                                gather_array(Vpart[itr][j],   $FG_COUNT,   V[itr][j],np);//V
send to all?
                                for i=0..$FG_COUNT-1
                                {
        execw0(i,NX,Apart[0][i],Apart[1][i],Apart[2][i],V[itr][j],wpart[itr][j][0][i])   @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                                }
                                gather_arrayDel(wpart[itr][j][0],                $FG_COUNT,
w[itr][j][0],np);
                                GaussSolve(L[0],L[1],L[2],N,1,w[itr][j][0],w[itr][j][1])   @
{request L[0],L[1],L[2],w[itr][j][0]; delete w[itr][j][0];};
                                GaussSolve(U[0],U[1],U[2],N,0,w[itr][j][1],w[itr][j][2])   @
{request U[0],U[1],U[2],w[itr][j][1]; delete w[itr][j][1];};
                                for i=0..$FG_COUNT-1
                        scatterVector(i,$FG_COUNT,ROWS,N,w[itr][j][2],wpart[itr][j][1][i]) @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); };//delete w[itr][j][2];};
                                for k=0..j
```

```
                {
                        for i=0..$FG_COUNT-1
                        {
        multipleVtow(i,NX,Vpart[itr][k][i],wpart[itr][j][k+1][i],Htemp[itr][j][k][i]) @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                        }
                normsum(Htemp[itr][j][k][0],Htemp[itr][j][k][1],spart[itr][j][k][0]) @
{request Htemp[itr][j][k][0], Htemp[itr][j][k][1]; delete Htemp[itr][j][k][0]; delete
Htemp[itr][j][k][1];};
                        for i=1..$FG_COUNT-2
                        {
                                normsum(spart[itr][j][k][i-
1],Htemp[itr][j][k][i+1],spart[itr][j][k][i])@ {request spart[itr][j][k][i-
1],Htemp[itr][j][k][i+1]; delete spart[itr][j][k][i-1]; delete Htemp[itr][j][k][i+1];};
                        }
                        executeHi(k,j,m,spart[itr][j][k][$FG_COUNT-
2],H[itr][j][k],H[itr][j][k+1]) @ {request m,spart[itr][j][k][$FG_COUNT-
2],H[itr][j][k]; delete spart[itr][j][k][$FG_COUNT-2]; delete H[itr][j][k];};

                        for i=0..$FG_COUNT-1
                        {
                                execw(j, k, i, m, NX, wpart[itr][j][k+1][i],
H[itr][j][k+1], Vpart[itr][k][i], wpart[itr][j][k+2][i]) @ {locator_cyclic:
$LOC1(i,$FG_COUNT,np); delete wpart[itr][j][k+1][i];};
                        }
                }
                norm_execute(wpart[itr][j][j+2],NX,norma[j],np);
                //print(norma[j][0]);
                executeHjplus(m,j,norma[j][0],H[itr][j][j+1],H[itr][j][j+2])
@ { request m,norma[j][0],H[itr][j][j+1]; delete H[itr][j][j+1];};
                for i=0..$FG_COUNT-1
                {

        initVpartj(j,i,m,NX,wpart[itr][j][j+2][i],H[itr][j][j+2],Vpart[itr][j+1][i]) @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); delete wpart[itr][j][j+2][i];};
                }

                for i=0..$FG_COUNT-1
                {

        reduceV(i,j+1,NX,Vp[itr][j][i],Vpart[itr][j+1][i],Vp[itr][j+1][i]) @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); delete Vp[itr][j][i];};
                }
```

```
        givensRotation(j,m,bnrm[0],H[itr][j][j+2],c[itr][j],s[itr][j],q[itr][j],H[itr][j+1][0
],c[itr][j+1],s[itr][j+1],q[itr][j+1],error[itr][j]) @ {locator_cyclic: 0; delete c[itr][j];
delete s[itr][j]; delete q[itr][j];};

                        if error[itr][j]<=$EPS {
                                init($RESTR+1, z[j+1]);
                        }
                        if error[itr][j]>$EPS init(j+1, z[j+1]);
                }
                intprint(K);
                calcR($RESTR,K,H[itr][K][0],R[itr]);// @ {delete betta[itr]; delete
H[itr][K][0];};
                for i=0..$FG_COUNT-1
                {
                        setNX(K,$FG_COUNT,MX[i])        @       {locator_cyclic:
$LOC1(i,$FG_COUNT,np);};
                        setROWS(K,$FG_COUNT,MX[i],MROWS[i])             @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                }
                for i=0..$FG_COUNT-1

        scatterMatrix(i,$FG_COUNT,MROWS[i],K,R[itr],Rpart[itr][i])              @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                for i=0..$FG_COUNT-1
                        calcypart(i,MX[i],K,Rpart[itr][i],q[itr][K],ypart[i])           @
{locator_cyclic: $LOC1(i,$FG_COUNT,np); delete Rpart[itr][i];};
                gather_arrayDel(ypart, $FG_COUNT, y,np);
                for i=0..$FG_COUNT-1 {
                        calcx0(i,NX,K,x0part[itr][i],Vp[itr][K][i],y,x0part[itr+1][i])
                                                        -->(Vp[itr][K][i])              @
{locator_cyclic: $LOC1(i,$FG_COUNT,np);};
                }
                gather_arrayDel(x0part[itr+1], $FG_COUNT, x0[itr+1],np);
                printime(error[itr][K-1], time1);
                if error[itr][K-1]<=$EPS {
                        init($ITMAX+1, x[itr+1]);
                }
                if error[itr][K-1]>$EPS init(itr+1, x[itr+1]);
                init(itr+1, x[itr+1]);
            }
        } @ {
                locator_replicating ROWS => all;
                locator_replicating N => all;
                locator_replicating NX => all;
```

```
                locator_replicating m => all;
                locator_cyclic bpart[i]=>i;
                locator_cyclic bnrm[i]=>i;
                locator_cyclic x0part[j][i]=>i;
                locator_cyclic Apart[j][i]=>i;
                locator_cyclic r0part[k][j][i]=>i;
                locator_cyclic betta[j][i]=>i;
                locator_cyclic Vpart[k][j][i]=>i;
                locator_cyclic Vp[k][j][i]=>i;
                locator_cyclic wpart[l][k][j][i]=>i;
                locator_cyclic Htemp[itr][j][k][i]=>i;
                locator_cyclic MROWS[i]=>i;
                locator_cyclic MX[i]=>i;
                locator_cyclic Rpart[itr][i]=>i;
                locator_cyclic ypart[i]=>i;
        }
        sub norm_execute(name arraypart, name lines, name norm,name np){
                df npart,spart;
                for i=0..$FG_COUNT-1
                {
                        normpart(i,lines,arraypart[i],npart[i]) @ {locator_cyclic:
$LOC1(i,$FG_COUNT,np);};
                }
                normsum(npart[0],npart[1],spart[0]) @ {locator_cyclic: 0; delete
npart[0]; delete npart[1];};
                for i=1..$FG_COUNT-2
                {
                        normsum(spart[i-1],npart[i+1],spart[i]) @ {locator_cyclic: 0;
delete spart[i-1]; delete npart[i+1];};
                }
                for i=0..$FG_COUNT-1
                {
                        setnorm(spart[$FG_COUNT-2],norm[i]) @ {locator_cyclic:
$LOC1(i,$FG_COUNT,np);};
                }
        } @ {
                locator_cyclic arraypart[i]=>i;
                locator_cyclic npart[i]=>i;
                locator_replicating lines => all;
                locator_cyclic norm[i]=>i;
        }
        sub findr0(name Apart, name bpart, name x0, name r0part, name NX, name
ROWS, name np){
                for i=0..$FG_COUNT-1
```

```
            {

        r0partexec(i,Apart[0][i],Apart[1][i],Apart[2][i],bpart[i],x0,NX,ROWS,
r0part[i]) @ {locator_cyclic: $LOC1(i,$FG_COUNT,np); request x0;};
            }
        } @ {
                    locator_cyclic r0part[i]=>i;
                    locator_replicating NX => all;
                    locator_replicating ROWS => all;
                    locator_cyclic Apart[j][i]=>i;
                    locator_cyclic bpart[i]=>i;
        }
        sub gather_array (name ar, int hs, name res_v, name np) { // hs for halfsize
              df sums;
            if hs == 2 {
                  cf red: gather_arr(hs,ar[0], ar[1], res_v) @ {locator_cyclic: 0;};
                  //cf copy: copy_v(sums,res_v);
             }
            if (hs>2){
               if (hs%2==0) {
                  for i=0..hs/2-1 {
                    cf azaza[i]: gather_arr(hs,ar[i*2], ar[i*2+1], sums[i]) @
{locator_cyclic: $LOC1(i*2,$FG_COUNT,np);};
                  }
                  gather_array(sums, hs/2,res_v,np);
               }
               if (hs%2!=0) {
                  for i=0..hs/2-1 {
                    cf azaza[i]: gather_arr(hs,ar[i*2], ar[i*2+1], sums[i]) @
{locator_cyclic: $LOC1(i*2,$FG_COUNT,np);};
                  }
                  cf copy_g[hs/2]: copy_v(ar[hs-1],sums[hs/2]);
                  gather_array(sums, hs/2+1,res_v,np);
               }
            }
        } @ {
                    locator_cyclic ar[i]=>i;
                    locator_cyclic sums[i]=>i;
        }

        sub gather_arrayDel (name ar, int hs, name res_v, name np) { // hs for halfsize
              df sums;
            if hs == 2 {
                  cf red: gather_arr(hs,ar[0], ar[1], res_v) @ {locator_cyclic: 0; delete
```

```
ar[0]; delete ar[1];};
                }
              if (hs>2){
                 if (hs%2==0) {
                   for i=0..hs/2-1 {
                     cf azaza[i]: gather_arr(hs,ar[i*2], ar[i*2+1], sums[i]) @
{locator_cyclic: $LOC1(i*2,$FG_COUNT,np); delete ar[i*2]; delete ar[i*2+1];};
                   }
                   gather_arrayDel(sums, hs/2,res_v,np);
                 }
                 if (hs%2!=0) {
                   for i=0..hs/2-1 {
                     cf azaza[i]: gather_arr(hs,ar[i*2], ar[i*2+1], sums[i]) @
{locator_cyclic: $LOC1(i*2,$FG_COUNT,np);};
                   }
                   cf copy_g[hs/2]: copy_v(ar[hs-1],sums[hs/2]);
                   gather_arrayDel(sums, hs/2+1,res_v,np);
                 }
               }
        } @ {
                    locator_cyclic ar[i]=>i;
                    locator_cyclic sums[i]=>i;
        }
```